



Using Diagrammatic Reasoning for Theorem Proving in a Continuous Domain

by

Daniel Winterstein

Supervisors

Professor Alan Bundy, Dr. Corin Gurr and Dr. Mateja Jamnik

Ph.D. in Informatics, The University of Edinburgh, 2004

Abstract

This project looks at using diagrammatic reasoning to prove mathematical theorems. The work is motivated by a need for theorem provers whose reasoning is readily intelligible to human beings. It should also have practical applications in mathematics teaching.

We focus on the continuous domain of *analysis* - a geometric subject, but one which is taught using a dry algebraic formalism which many students find hard. The geometric nature of the domain makes it suitable for a diagram-based approach. However it is a difficult domain, and there are several problems, including handling alternating quantifiers, sequences and generalisation. We developed representations and reasoning methods to solve these. Our diagram logic isn't complete, but does cover a reasonable range of theorems. It utilises computers to extend diagrammatic reasoning in new directions – including using animation.

This work is tested for soundness, and evaluated empirically for ease of use. We demonstrate that computerised diagrammatic theorem proving is not only possible in the domain of real analysis, but that students perform better using it than with an equivalent algebraic computer system.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

(Daniel Winterstein)

Acknowledgements

I would like to thank Professor Alan Bundy, Dr. Corin Gurr and Dr. Mateja Jamnik, my supervisors, for their suggestions and support during this research. Dr. Gurr's patience and assistance with draft write-ups was particularly invaluable. My examiners, Professor John Howse and Dr. Anthony Maciocia, were lovely, and I am grateful to them for their time and help in producing the finished thesis.

I have also received help from other academics. The Dream Group (a.k.a. the Edinburgh Mathematical Reasoning Group) are all great people, and it has been an honour to work among them. Dr. Jacques Fleuriot and Professor Keith Stenning gave me valuable feedback, and Dr. Derek Arthur in the Mathematics department helped arrange the experiments I conducted. I am also grateful to Professor Bruno Buchberger for his hospitality, and Dr. Tom Körner, an inspiring mathematics teacher and genuine eccentric, who first lit my interest in pure mathematics.

I am grateful to my family and friends for salvaging as much of my sanity as they could. Without their patience and kindness this work would not have succeeded. Special thanks go to Mark and my parents for generously volunteering to proof-read, and to all those in Edinburgh who had to put up with my sulks, tantrums, mood swings, wild ravings and neglect of washing up duties: Alison, Becca, Joe, Paul, Seb, Tingy & Voon.

This work was conducted on an EPSRC studentship, with additional funding from Calceumus. I am grateful to both organisations for their support.

“Of times Archimedes' servants got him against his will to the baths, to wash and anoint him, and yet being there, he would ever be drawing out of the geometrical figures, even in the very embers of the chimney. And while they were anointing of him with oils and sweet savours, with his fingers he drew lines upon his naked body, so far was he taken from himself, and brought into ecstasy or trance, with the delight he had in the study of geometry.” – Plutarch

“Soldier, stand away from my diagram.”

– Archimedes' last words (apocryphal)

Table of Contents

1 Introduction.....	17
1.1 Real analysis.....	20
1.2 Motivation: overview.....	21
1.3 Project aims.....	21
1.4 Thesis overview.....	22
2 Survey of the Field.....	25
2.1 Theoretical discussions.....	25
2.1.1 What is a diagram?.....	25
2.1.2 What are the advantages of diagrammatic reasoning?.....	26
2.1.3 Can diagrammatic proof be formalised?.....	29
2.2 Representation schemes.....	31
2.2.1 Direct and indirect representations.....	31
2.2.2 Diagrams and geometry.....	32
2.2.3 Representing quantifiers.....	33
2.3 Existing diagrammatic reasoning systems.....	33
2.3.1 Diagram guided theorem provers.....	33
2.3.2 Diagram interpretation theorem provers.....	35
2.3.3 Dynamic reasoning systems.....	37
2.4 Summary.....	40
3 Diagrammatic Reasoning: Problems & Challenges.....	43
3.1 Problems in diagrammatic reasoning.....	44
3.1.1 Impossible drawings.....	44
3.1.2 Optical illusions.....	45
3.1.3 Roughness of drawing.....	46
3.1.4 Drawing mistakes.....	51
3.1.5 Ambiguity.....	51
3.2 Algebraic pitfalls.....	52

3.2.1 Roughness of representation.....	53
3.2.2 Limitations of what can be represented.....	54
3.2.3 Textual illusions.....	54
3.2.4 Ambiguity.....	54
3.2.5 Hidden assumptions.....	55
3.3 Challenges in diagram logic.....	55
3.3.1 Disjunction.....	55
3.3.2 Negation.....	56
3.3.3 Contradiction.....	56
3.3.4 Extending to new concepts and domains.....	56
3.3.5 Quantifiers.....	57
3.4 Summary.....	58
4 A Diagram Logic for Analysis Proofs.....	59
4.1 An example proof: $f(x) = 1/x$ is continuous on $(0, \infty)$	60
4.1.1 Diagrammatic proof of the theorem.....	60
4.1.2 Algebraic proof of the theorem.....	65
4.1.3 Summary of example proof.....	66
4.2 Dynamic diagram logic.....	66
4.2.1 Representation of relations.....	67
4.2.2 The structure of the proof.....	67
4.2.3 Generalisation in diagrammatic proof.....	67
4.2.4 Introducing redraw rules.....	69
4.2.5 Sometimes a diagram says too much.....	71
4.2.6 Using branching for disjunction.....	71
4.2.7 Quantifiers.....	72
4.2.8 Aspects of DDL not illustrated in §4.1.....	78
4.3 Formalising DDL.....	82
4.3.1 Notation.....	82
4.3.2 Diagrams.....	83
4.3.3 Redraw rules.....	89
4.3.4 Meta rules.....	97
4.3.5 Counterexamples.....	98
4.4 Using dynamic diagram logic for analysis.....	99
4.4.1 Scope of DDLA.....	100

4.4.2 Design issues in DDLA.....	100
4.4.3 Objects.....	101
4.4.4 Constants.....	104
4.4.5 Representations.....	104
4.4.6 Implicit inferences.....	114
4.4.7 Emergent objects.....	116
4.4.8 Rules.....	117
4.5 Conclusion.....	119
5 Soundness.....	121
5.1 Background discussions.....	122
5.1.1 Assumptions in DDLA.....	122
5.1.2 Diagram semantics (c.f. §5.2.1).....	126
5.1.3 From soundness in DDLA to soundness in standard analysis.....	129
5.2 Formal definitions and results.....	130
5.2.1 Diagram semantics (c.f. §5.1.2).....	130
5.2.2 Soundness of the inference mechanism.....	134
5.2.3 Summary.....	143
5.3 The subset relation.....	143
5.3.1 Representing $A \subset B$	144
5.3.2 Transitivity.....	147
5.3.3 Diagram-model link.....	148
5.4 Examples of showing specific rules are sound.....	154
5.4.1 Simple rule: open set definition.....	154
5.4.2 Branch rule: set union definition.....	155
5.4.3 Animated rule: recognising $Y \subset X$	156
5.5 Summary.....	157
6 Evaluation.....	159
6.1 Evaluation criteria.....	159
6.1.1 Project aims.....	159
6.1.2 Ways of evaluating.....	160
6.1.3 Hypothesis for evaluation.....	161
6.2 Mathematical evaluation.....	162
6.2.1 Soundness.....	162
6.2.2 Range.....	162

6.3 Computer implementation: Dr.Doodle.....	166
6.3.1 Motivation.....	166
6.3.2 DDLA and Dr.Doodle.....	167
6.3.3 System design.....	168
6.3.4 Algebraic mode.....	176
6.3.5 Testing.....	179
6.3.6 Summary.....	179
6.4 Empirical evaluation.....	180
6.4.1 Constraints.....	180
6.4.2 Test subjects.....	180
6.4.3 Results.....	183
6.4.4 Future work.....	185
6.4.5 Conclusion.....	186
6.5 Summary.....	187
7 Related Work.....	189
7.1 Diamond.....	190
7.1.1 Similarities.....	190
7.1.2 Differences.....	191
7.2 Spider diagrams & constraint diagrams.....	192
7.2.1 Emergent objects.....	194
7.2.2 Representing disjunction.....	194
7.2.3 Representing negatives.....	195
7.2.4 Representing implication.....	196
7.2.5 Representing quantifiers.....	196
7.2.6 Domain flexibility.....	197
7.3 Summary.....	197
8 Future work.....	199
8.1 Improving Dr.Doodle.....	200
8.1.1 Improving the user interface	200
8.1.2 Representation.....	200
8.1.3 Full DDLA compliance.....	201
8.1.4 Automated drawing.....	201
8.1.5 Automated reasoning.....	203
8.2 Extending DDLA.....	203

8.2.1 Extending the representation scheme.....	203
8.2.2 Using outside reasoning systems.....	205
8.2.3 More accuracy.....	205
8.2.4 Not rules.....	206
8.2.5 Sequences.....	206
8.2.6 Differentiation.....	207
8.2.7 Integration.....	210
8.3 Summary.....	210
9 Conclusion.....	213
9.1 Aims.....	213
9.2 Our work.....	214
9.2.1 Exploration.....	214
9.2.2 Formalisation.....	214
9.2.3 Implementation.....	215
9.3 Assessment.....	216
9.4 Where next?.....	217
9.4.1 Extensions to DDLA.....	217
9.4.2 Other applications.....	217
9.5 Closing thoughts.....	217
10 Glossary of Terms.....	221
10.1 Notational conventions.....	221
10.2 Terms defined in this project.....	221
11 References.....	225
Appendix A: Rule-set for DDLA.....	231
Appendix B: Sequence Reasoning Example.....	249
Appendix C: Some Example Proofs.....	253
14.1 Nested balls lemma.....	253
14.2 Open set union.....	255
14.3 A theorem about continuity.....	259

Index of Illustrations

Figure 1.1. This geometric proof of Pythagoras' Theorem is a classic example of diagrammatic reasoning.....	17
Figure 2.1. A Schubert diagram for “Several children are in the playground. Most of them are playing in the sandbox”.....	28
Figure 2.2. We are unsure what this Schubert diagram means.....	28
Figure 2.3. Triangles suggesting different generalisations (from left to right: all triangles, all right angled triangles and all right-angled isoceles triangles).....	31
Figure 2.4. Diagram for proving the Schröder-Bernstein theorem ¹⁶ with “&”/Grover.....	34
Figure 2.5. A HyperProof diagram.....	36
Figure 2.6. A proof in Hammer's diagrammatic logic of $x \in A \cup B, B \subset C \Rightarrow x \in A \cup C$	39
Figure 2.7. Screenshot of Mondrian. The 'dominoes' on the left are drawing tools, with drawing performed in the space to the right.....	40
Figure 3.1. The physically impossible Penrose Triangle.....	44
Figure 3.2. None of these objects are physically possible.....	45
Figure 3.3. An optical illusion: the horizontal lines are parallel.....	45
Figure 3.4. An attempt to plot \mathbb{R} and \mathbb{Q} with points plotted as dots of radius 1 printing pt...	46
Figure 3.5. An example of Jamnik's 'topological representation'.....	47
Figure 3.6. Rearranging the 4 shaded blocks seems to change their total area.....	50
Figure 3.7. Young or old? (hint: the young woman's chin is the old woman's nose).....	52
Figure 3.8. Ambiguity in diagrams.....	52
Figure 3.9. A textual illusion.....	54
Figure 3.10. A fallacious proof that $1=2$	55
Figure 4.1. Diagrammatic proof that $f(x)$ is a surjective function (i.e. $\forall y \exists x. f(x)=y$).....	60
Figure 4.2. Diagrammatic definition for $f(x) = 1/x$	61
Figure 4.3. Diagrammatic proof that $f(x)=1/x$ is a decreasing function.....	62
Figure 4.4. Redraw rule defining continuity.....	63

Figure 4.5. Diagrammatic proof that f surjective and decreasing $\Rightarrow f$ continuous.....	64
Figure 4.6. Differing approaches to proving that $f(x)=1/x$ is well defined.....	67
Figure 4.7. Definition 4.2.4.1 as a redraw rule.....	70
Figure 4.8. The $x \in X \cup Y \Rightarrow x \in X$ or $x \in Y$ branch rule with (left) and without (right) modifying the underlying model.....	72
Figure 4.9. Definition 4.2.4.2 as an animated redraw rule.....	73
Figure 4.10. Sketch proof for “ $B_r(x)$ is open”.....	76
Figure 4.11. Antecedent matching with strict and flexible transitions.....	77
Figure 4.12. Emergent objects in Pythagoras' theorem.....	79
Figure 4.13. A counterexample to $A \subset X \cup Y \Rightarrow A \subset X$ or $A \subset Y$	79
Figure 4.14. Equivalent triangles?.....	88
Figure 4.15. Example illustrating program matching.....	93
Figure 4.16. Example of adaptive matching: The right-angled condition is 'discovered' and added to the reasoning program, causing a pruning cut to remove part of the reasoning program.....	96
Figure 4.17. Representation of the merge rule in Dr Doodle.....	98
Figure 4.18. Example representations for object types in DDLA (taken from Dr Doodle screenshots).....	101
Figure 4.19. Using line segments to represent numbers.....	105
Figure 4.20. $(a+b)^2$ represented with line-segments and area blocks, giving the free inference $(a+b)^2 = a^2 + 2ab + b^2$	106
Figure 4.21. Representing function and inverse relations when working with 1D spaces...	111
Figure 4.22. It is possible for a diagram to implicitly represent $x=y$ without stating $f(x)=f(y)$. By contrast, it is not possible to implicitly represent $x \in Z$, $Z \subset X$ without also representing $x \in X$	115
Figure 4.23. Example of using emergent sets in DDLA.....	116
Figure 4.24. Function application for points rule.....	117
Figure 4.25. Rule for recognising $Y \subset X$	118
Figure 4.26. Set union definition: $x \in X \cup Y \Rightarrow x \in X$ or $x \in Y$	118
Figure 5.1. The same diagram drawn at 3 different resolutions.....	125
Figure 5.2. Diagrams and mappings used in Lemma 4.....	136
Figure 5.3. Example of $\text{implicit}(a \in B)$ but $a \notin B$	148
Figure 5.4. Disconnected M.....	150

Figure 5.5. $ M > 2$	151
Figure 5.6. Example of $a \notin B$ but $\mathbb{D}(a) \subset \text{interior}(\mathbb{D}(B))$	152
Figure 5.7. Open set definition.....	154
Figure 5.8. Set union definition.....	155
Figure 5.9. Subset definition.....	156
Figure 6.1. The working window.....	172
Figure 6.2. The theorem window.....	173
Figure 6.3. The redraw rules window.....	174
Figure 6.4. The lesson browser.....	174
Figure 6.5. The working window in algebraic mode.....	177
Figure 6.6. A proof-in-progress in algebraic mode, showing an automated inference.....	179
Figure 7.1. A Diamond-style proof.....	190
Figure 7.2. A spider diagram.....	193
Figure 7.3. A constraint diagram.....	193
Figure 7.4. Expressing "If X is an open set..." using a constraint diagram (left) and a DDLA rule (right).....	197
Figure 8.1. Example of a diagrammatic object label (exerted from Figure 4.10).....	203
Figure 8.2. Defining uniform continuity using an animated consequent.....	204
Figure 8.3. An approximation function with a constant error margin.....	208
Figure 8.4. Defining differentiability using error margins.....	209
Figure 8.5. Defining continuity using error margins.....	209
Figure 8.6. Diagrammatic definition for integrability.....	210
Figure 8.7. Diagrammatic proof that f continuous, X closed & bounded $\Rightarrow f$ integrable on X	211

Appendices

Figure 13.1. Closed X and X^c	249
Figure 13.2. Constructing example points in a sequence.....	250
Figure 13.3. Sequence generator rule.....	251
Figure 13.4. Proving the sequence converges to y	251
Figure 14.1. Lemma statement.....	253
Figure 14.2-7. Diagrammatic proof.....	254-255
Figure 14.8. Theorem statement.....	256
Figure 14.9. The structure of the proof program.....	256

Figure 14.10-15. Diagrammatic proof.....	256-258
Figure 14.16. Dr.Doodle screenshot showing the verified proof.....	259
Figure 14.17. Theorem statement.....	260
Figure 14.18-27. Diagrammatic proof.....	260-263

1 Introduction

Diagrams are commonly used in virtually all areas of representation and reasoning. In particular, they are invaluable in mathematics texts. They are used in a variety of ways, including to give examples showing why a theorem is true, to give counter-examples, or to explain the structure of a proof. More rarely, diagrams can be used to prove a theorem outright. Insight is often more clearly perceived in these diagrammatic proofs than in the corresponding algebraic proofs. As Nelsen observes in “Proof without Words”, “in English ‘to see’ is often ‘to understand’” [37]. This project looks at using diagrammatic reasoning to prove mathematical theorems. This chapter gives a brief overview of the motivation for this project, the work achieved, and how it advances the field.

The aims of the project were to develop a way of reasoning with diagrams suitable for doing analysis proofs, and demonstrate its potential usefulness by building a prototype teaching tool based on it. This was motivated by the belief that diagrammatic reasoning is more intuitive for some domains.

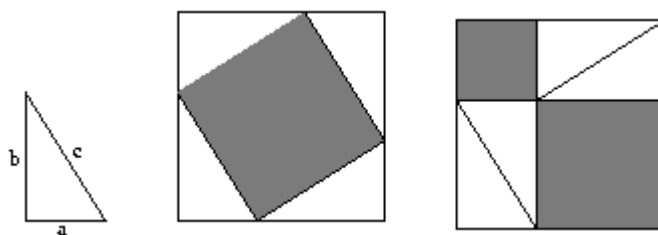


Figure 1.1. This geometric proof of Pythagoras' Theorem is a classic example of diagrammatic reasoning.

Although modern mathematicians give algebraic proofs, this work is frequently driven by a geometric understanding. Barker-Plummer claims visualisation is a key aspect “not just of mathematical learning but also [of] mathematical discovery”, and diagrams “play an essential role” in this [1]. It is not surprising that geometry (and geometric reasoning) was the original form of mathematics. For example, Pythagoras' Theorem was proved circa

500BC. The Pythagoreans' proof was lost, and as with anything relating to Pythagoras, it is impossible to know just what was done, when and by whom. However, his proof would almost certainly have been geometric [39]. The elegant proof in Figure 1.1 is due to an unknown Chinese mathematician writing circa 200BC [37]. By comparison, algebra is a recent invention, usually attributed to al-Khwarizmi in 830AD.¹ The modern algebraic formalism is barely a hundred years old, the result of the axiomatisation project of Hilbert, Frege, Russell *et al.* A side-effect of that great project is that diagrams have fallen out of favour as acceptable methods of proof. Only algebra is regarded as formal. The current monopoly of algebraic formal mathematics is summed up by Tennant: “[the diagram] is only an heuristic... it has no proper place in the proof”[2].

This insistence on algebraic formalism is a curious position, as Barker-Plummer observes: “most mathematicians deny that diagrams have any formal status, but on the other hand, diagrams are ubiquitous in mathematics texts” [2]. Moreover their second class status may well be detrimental, as Eisenberg and Dreyfus assert in a study on visualisation in mathematics: “it is our contention that students (and oftentimes their teachers too) cannot do many of the [sample problems] because... the visual characteristics of the problems were not even considered” [8]. In the light of this, it is interesting to note that in al-Khwarizmi's work, the use of algebra is justified with geometric proofs [39].

Given the advantages of diagrams and their widespread day-to-day use, it might seem strange that the field has been so neglected, both in mathematical logic and in practical computer work.

Fallacious diagrammatic proofs cast doubt on the validity of diagrammatic proofs. Worse than this was the discovery of functions with completely unintuitive behaviour, such as space-filling curves. Diagrammatic reasoning came to be considered unreliable and not rigorous enough for mathematical proof. We believe that computers can be used to eliminate these problems.² Combining the computer's 'immunity' to optical illusions with the rigid structure imposed by automated reasoning techniques should allow us to give guarantees of soundness in diagrammatic proof.

1 One could argue it began with Diophantus circa 250AD, but this does not affect our point.

2 After all, human reasoning with algebra can also have errors. c.f. [32] for examples of some subtle ones.

There are also pragmatic reasons why diagrammatic reasoning is not as well developed as its sentential³ counterpart. Although formal logic was invented as a branch of pure mathematics, it found very successful applications in the design of computers and programs. This drove further development of the subject. Diagrams are mainly important to understanding – and they could still be used for this in mathematics without a formal theory. Thus there was no equivalent motivation for developing diagrammatic reasoning.

Computers are now common-place, and programmers can no longer expect their users to be experts. This, combined with increases in computing power which have made the overhead of graphics much less prohibitive, means that for most 'desktop' software, HCI⁴ issues are now more important than straight computational ones. Comparing current day GUIs⁵ to the rich visual representations used in non-computer based work – such as circuit diagrams, flow charts or architectural floor plans – shows the potential for improvement. However, although 'paper' diagrams are currently more versatile and sophisticated, computers have a lot to offer. Computer graphics tools give fast and accurate drawing, which could make complex and intricate diagrams much easier to use.

There is also the exciting possibility of developing diagrams in new ways. Diagrams on paper are necessarily static. If we consider the very real differences between text and hypertext, we see that diagrammatic reasoning on computers needn't be just a straight conversion of diagrammatic reasoning on paper. Possible directions include the use of animated or interactive diagrams. A theoretical underpinning for such work is clearly desirable.

Developing a diagrammatic logic for theorem proving is therefore interesting on two fronts: as a useful tool for computer-aided mathematics, and to investigate the nature of diagrammatic reasoning (where the rigour required for doing mathematics will force a thorough investigation of the mechanics of such reasoning).

3 *Sentential*: involving sentences. Synonymous here with *algebraic* or *textual*.

4 HCI: Human Computer Interaction

5 GUI: Graphical User Interface

1.1 Real analysis

The domain we will investigate is that of theorem proving in real analysis and Euclidean space analysis, and in particular, those theorems which involve a continuum of different cases. For those unfamiliar with analysis, [27] provides an excellent introduction. Ours is the first attempt to apply a visual approach to this domain since Maclaurin tried back in the 18th century to put the calculus on a rigorous basis using geometric reasoning. He failed [39]. We are proposing to re-attempt that undertaking, but with the vast advantage of 250 years of mathematical and computer development to draw on.

A continuous domain was chosen because – in the right circumstances – a continuum of cases can be understood as if they were a single case. This is because there is often a smooth transition from one case to another. It suggests that representing them with a single diagram will be viable.

Historically, Euclidean space analysis is a generalisation of real analysis, which was developed to justify the use of calculus. \mathbb{R}^3 was meant to be the real world, and the definitions were supposed to capture how the universe works. Arguably most of the work went into coming up with the right definitions. As the universe is a geometric place, it is not surprising that many of the concepts are best understood geometrically. The traditional algebraic formalism is often confusing to students meeting it for the first time [53]. Even great mathematicians such as Cauchy have made mistakes in this subject [32][28].

Therefore, this domain should give a good demonstration of these ideas, letting us replace complex algebraic formulations with the geometric concepts they represent.

The domain includes \mathbb{R} (the line) and \mathbb{R}^2 (the plane), which means that we can, in places, draw examples of the objects we are reasoning about.⁶ Barwise & Etchemendy argue that the power of diagrammatic reasoning comes from allowing representations that share structure with the target domain [3]. Allowing objects to represent themselves – the extreme form of sharing structure – is therefore very promising.

⁶ As we examine in §3.1.3, such drawings are really *approximations* of examples.

1.2 Motivation: overview

- 1) Investigate a new form of logic.

Research is required to find out how diagrammatic logics can work.

- 2) Develop diagrammatic reasoning in new directions

The use of computers opens up new realms of diagrammatic reasoning - such as animated diagrams and interactive diagrams.

- 3) Improve Human-Computer Interaction in mathematical tools

Since diagrams are such powerful tools in human communication, they have a rich potential for aiding HCI.

- 4) Develop new teaching aids for mathematics

People find analysis and related subjects very hard to understand. We think this is largely due to the dry algebraic nature of the formalism which hides what are "fairly simple" geometric ideas.

The assumption underlying this project is that, for some domains, diagrammatic reasoning is easier to understand for a significant number of people.

1.3 Project aims

The overall aim of this project is to investigate the potential for applying a diagrammatic approach to mechanised reasoning. Since there has been little research to date in this area, and none in the domain we focused on, the first stage of the project was exploratory. Its aim was to develop sufficiently powerful diagrammatic techniques to tackle analysis problems. The techniques developed should then have a practical application in mathematics teaching, where, we hope, they will complement conventional methods (although any such application is beyond the scope of this project).

Hence this project has two goals:

- 1) To develop a formal logic that uses diagrammatic reasoning to solve problems in real analysis.
- 2) To make a case for this logic having advantages over the conventional algebraic approach.

1.4 Thesis overview

We now look at the layout of this thesis, giving a chapter-by-chapter overview.

1) *Introduction (this chapter)*

Describes and motivates the problem we tackled.

2) *Literature Survey*

Gives a broad survey of the field and where this work fits in. Diagrammatic reasoning research is still in its infancy, and this work is new in several respects. The domain we look at has not been tackled before diagrammatically. Although some of the problems involved have been analysed already, the domain we consider presents considerable challenges, requiring new solutions.

3) *Diagrammatic Reasoning: Problems & Challenges*

Discusses the problems presented by diagrammatic reasoning, including roughness of drawing, ambiguities, handling quantifiers, disjunctions and generalisation.

4) *A Diagram Logic for Analysis*

This chapter forms the 'core' of this thesis. It gives examples of our reasoning style. It then sets out the methods used for representation and reasoning, and explains how they work. This is done both informally and formally.

5) *Soundness*

This chapter shows that the logic described in the previous chapter is sound. This is based on a two-level analysis. Firstly we look at the link between physical diagrams and abstract diagram-descriptions, then we consider the soundness of rule application and diagrammatic proof at an abstract level. A conversion function is given for diagram rules, allowing us to compare our rules with the standard definitions.

6) *Evaluation*

This chapter analyses the capabilities and limitations of the logic. We evaluated our work against the hypothesis: “Diagrammatic proofs are possible for analysis problems, and may be easier (in some sense) than algebraic reasoning”. This involves evaluating coverage of the domain and ease of use. An empirical study was conducted to evaluate

the potential of this work for teaching mathematics. This chapter also describes the interactive theorem prover, called **Dr.Doodle**, built to implement our ideas.

7) Related Work

This project is then compared with similar research: Jamnik's **DIAMOND** system, and Howse *et al*'s work with Spider Diagrams.

8) Future Work

This chapter looks at how this work can be extended, and the potential benefits of doing so. It focuses on extensions within our target domain (such as reasoning about sequences), but also considers how ideas from this project might be applied to other areas (e.g. interactive program generation).

9) Conclusion

This chapter draws together the various threads of the project into a glorious tapestry, and considers the meaning of this work from a wider perspective. It delivers a devastating indictment of the current corrupt and degenerate state of work in the field, but concludes with a message of hope: a shining vision of a future built on brotherhood, justice and diagrammatic reasoning.

- *Appendix A: Rule set with soundness proofs*
- *Appendix B: An example of diagrammatic reasoning with sequences*
- *Appendix C: Some example proofs*
- *CD-ROM*

This provides illustrations and hands-on examples for chapters 4 and 6. It contains a version of the **Dr.Doodle** software, and an electronic copy of this document.

2 Survey of the Field

This chapter gives a broad survey of the field and where our work fits in. It covers theoretical issues, different representation schemes and existing diagrammatic reasoning systems.

We examine attitudes towards diagrammatic proof. For a long time, diagrams were considered unsuitable for rigorous mathematical proof. However, recent work on formalising diagrammatic systems has changed opinions – though the issue remains slightly controversial.

Within the diagrammatic reasoning community, it is often taken as self-evident that diagrams are easier and/or better for people to reason with than text. However diagrammatic reasoning is not always superior to sentential reasoning. The best approach depends on both the problem and – which is often overlooked – the person.

Diagrammatic reasoning research is still very much in its infancy, and there are many unanswered questions. In particular, there is no comparable work for the domain of mathematical analysis which we tackle. However diagrammatic reasoning is attracting increasing attention, and there are a considerable number of diagrammatic reasoning systems, including: the Geometry Machine [55], “&”/GROVER [2], HYPERPROOF [16] and DIAMOND [24], and the work on Constraint Diagrams [11].

2.1 Theoretical discussions

2.1.1 What is a diagram?

There are numerous different definitions in the literature for what counts as a diagram. Fortunately, it seems research in the field has moved on from this semantic chimera. Whilst a 1995 round up of the field contains a host of discussions on this question, the Diagrams 2002 conference did not mention it once [62][64]. For the purpose of this thesis, we will

take a broad view, considering diagrams to include heterogeneous representations that mix visual elements and text.

2.1.2 What are the advantages of diagrammatic reasoning?

In the introduction we claimed that diagrams are more intuitive and easier to understand than algebra. This seems a straightforward claim, and certainly there are many areas where this is uncontroversial. For example, we would not find many electricians who describe circuits without diagrams. Over 50% of the cortex in primates is devoted to visual processing [43], and so it is not surprising that people find visual representations and operations natural to use. Some of the advantages that have been claimed for diagrams are:

- Diagrams implicitly represent complex relationships, alleviating problems that occur when the number of elements and connections exceeds short term memory [21].
- Diagrams make the abstract concrete [21].
- Diagrams “help learners build runnable mental models” [33].
- “Diagrams automatically support a large number of perceptual inferences, which are extremely easy for humans.” [48].

However there is evidence that diagrams are not the best method for all students. In an experiment on learning about mechanical systems, Heiser & Tversky found that subjects with low mechanical ability performed better with textual descriptions than diagrams [18]. In the domain of teaching logic, Stenning *et al*'s studies of HyperProof show that some students perform better using HyperProof without diagrams [51]. They conjecture that students use one of two styles of reasoning: spatial or algebraic. They link these with a serialist approach and the use of algebra, or a holistic approach and the use of diagrams. Teaching a student using the wrong type of method results in poor performance. A study of syllogistic reasoning, comparing the use of Euler circles against a sentential technique, found a similar pattern [38].

The evidence is not conclusive yet. The results of some of these experiments were “not significantly correlated” [51]. It is also very hard to separate out the ability we wish to test from other mental processes that could be involved. For example, Stenning *et al* use the 'Paper Folding Test' to measure spatial reasoning ability, but concede that it may in fact be measuring the ability to translate verbal problems into spatial form [38]. Finally, their work

has so far compared only a very small sample of reasoning methods (HyperProof with and without graphics, and Euler Circles versus Natural Deduction). It is possible that the differences between student reasoning styles are more complex. There are other aspects of representations that can affect reasoning style besides the diagrammatic/algebraic distinction. For example, Venn Diagrams and Euler Circles are two diagrammatic methods for the same domain which give rise to quite different reasoning styles.⁷ A study of student learning patterns between these would be of interest here. There may also be effects associated with learning *multiple* representation methods, rather than any single method.

What research in this area definitely shows is that there are wide variations between students. These variations suit different styles of reasoning, which mathematics teaching should take into account [53]. In the case considered in this project – that of teaching mathematical analysis – current teaching practices generally ignore these variations. In spite of the fact that analysis concepts are mostly geometric, the normal approach is heavily biased towards algebraic reasoning. This is because experience has taught that the subject requires rigorous care. Conventionally, only algebra is regarded as being sufficiently rigorous.

Good and bad diagrams

Just as some sentences work better than others,⁸ some diagrams are better than others. What makes a good diagram is an interesting and difficult question.

It is perhaps illustrative to give an example of how poor diagrams can come to be developed. Many diagram systems are quite limited in the scope of what they can represent. The easiest way to extend a diagrammatic representation or reasoning scheme is by introducing new notation. This can produce systems as powerful as any logic. However, it generally leads to more complex diagrams, and great care is required if these are to retain their intuitive feel. For example, in 1976 Schubert, starting from semantic nets, developed (by adding more and more notation) a diagrammatic representation that is as expressive as modal lambda calculus (see Figure 2.1 and Figure 2.2) [44]. Unfortunately, the resulting diagrams are extremely difficult to read and, to the best of our knowledge, were not generally used.

⁷ Mistakes with Euler Circles will probably come from missing cases, whilst with Venn Diagrams, extracting the conclusion would seem to be the hard part.

⁸ See text for examples.

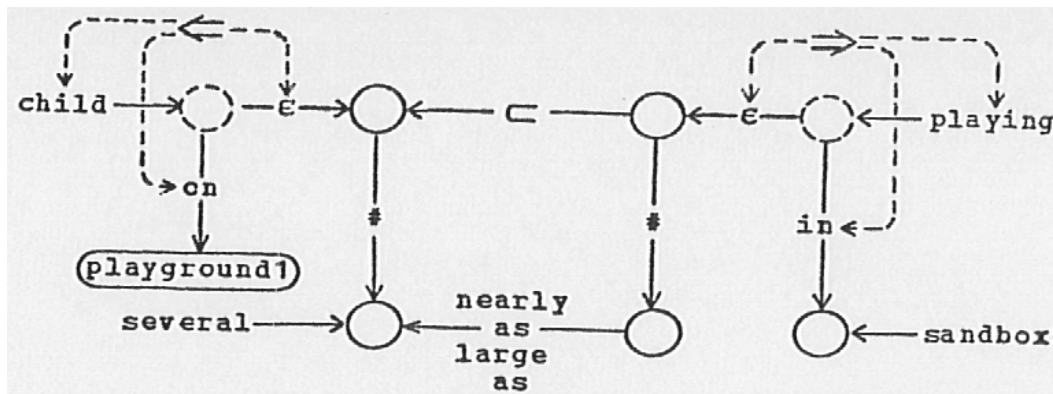


Figure 2.1. A Schubert diagram for “Several children are in the playground. Most of them are playing in the sandbox”

Horn explores diagram design, developing various guidelines for good design [21]. He finds that there is “very little rigorous research” on the subject, although in places he draws on ideas from psychology studies. His suggestions include using colour, similarity, proximity, continuation and enclosed regions to convey information. Graphical features such as contrast or size can also be used to focus attention on key parts of a diagram, but it is important not to overload a diagram with too many such focus points.

When designing a new diagrammatic representation scheme there are several factors that should be taken into account, such as who will use it (laymen or experts) and what role it will play (e.g. active reasoning, or communicating data). There are trade-offs between clarity, expressive power and the learning curve for users.

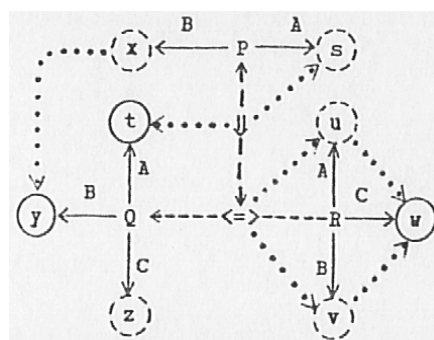


Figure 2.2. We are unsure what this Schubert diagram means.

2.1.3 Can diagrammatic proof be formalised?

Although diagrams are ubiquitous in mathematics texts, many logicians deny that diagrams have any formal status [2]. This can perhaps be attributed to the success of Hilbert, Frege, Russell *et al* in formalising algebraic methods, and the lack of an equivalent body of work for diagrams.⁹ In [13], Greaves argues that “the familiar prejudice against diagrammatic inference in logic and geometry owes more to history and philosophical context than to any technical incompatibility with modern theories of axiomatic systems”.

Shin's work on formalising Venn diagrams seems to have shifted opinion on this [46]. Other formalisations of diagrammatic systems have followed, e.g. [22] or [14]. By treating diagram objects as objects in \mathbb{R}^2 , properties about their basic behaviour can be proved (namely that curves can be used to define regions, and the transitivity of the inside relation for such regions), and from this a formal system can be built up with well-defined syntax and semantics. Properties of the representation (e.g. completeness within a domain) and reasoning rules (e.g. soundness) can then be proved in the same way as for sentential logics. There remains the question of how valid it is to equate curves in a diagram with curves in \mathbb{R}^2 . The literature typically ignores this issue, tacitly assuming that the translation from paper to \mathbb{R}^2 does not cause problems.¹⁰ As we explore in §3.1.3 and §5.3, this issue does, in fact, create some theoretical difficulties, and could potentially lead to mistakes. In particular, inside/outside relations can change when a curve in \mathbb{R}^2 is physically drawn. However, these problems only affect extreme cases, and we will show in §5.3 that this issue can be dealt with in a rigorous way.

Proving properties of a diagrammatic reasoning system from the geometry of the drawing plane is an arduous task though. Moreover, Venn diagrams use only closed curves, regions and points, which are relatively simple to treat mathematically. The addition of more complex representations (e.g. with dotted lines, arrows, etc.) would make this standard of proof much harder. Hence it is not surprising that other work on the soundness of

⁹ Interestingly, although Hilbert's “Foundations of Geometry” turned Euclidean geometry into a formal algebraic logic, some of his proofs *do* involve diagrams, in that there are necessary conditions which are only ever stated in the diagrams accompanying the proofs [34]. Also, Frege's original notation had diagrammatic features.

¹⁰ Several formalisations cite the beneficent presence of the Jordan Curve Theorem when justifying a diagrammatic system (e.g. [15]). Quite how the rather precise and technical theorem “Continuous injective images of $[0,1]/\{0=1\}$ into $\overline{\mathbb{Q}} \times \overline{\mathbb{Q}}$ are homeomorphic to $\{x \in \overline{\mathbb{Q}} : |x|=1\}$ ” relates to actual diagrams is usually left as an exercise for the reader. Note that the Jordan Curve Theorem does not necessarily hold for surfaces that approximate \mathbb{R}^2 (e.g. $\mathbb{Q} \times \mathbb{Q}$).

diagrammatic reasoning (e.g. [14]) has typically assumed that diagrams can be reliably and properly parsed, and shown soundness at the interpretation level.

Recent critics of computerised diagrammatic reasoning include Penrose, who claimed it is not possible to automate certain diagrammatic proofs – specifically number theory proofs involving intuitive leaps to generalise from the diagram to the proof. However Jamnik's DIAMOND system is capable of proving such theorems¹¹ [24], so these objections no longer seem valid.

Generalisation

More recently, Hayes raised questions about whether we can systematically generalise diagrammatic proofs of geometry theorems [17]. He showed that a diagrammatic proof can give rise to several valid generalisations. The different generalisations depend upon which aspects of the diagram are judged to be important. Clearly this phenomenon presents a problem for a diagram logic.

However the process of formalising a logic involves setting out an explicit method for generalisation, which removes this problem (c.f. §4.2.3). Essentially, formalisation restricts how diagrammatic reasoning is meant to function. Removing the problem in this way though raises the possibility of creating a logic that is sound but misleading. That is, although only one generalisation would be valid under the rules of the logic, users might be tempted by their intuition to assume a different, invalid, generalisation.

Hayes argues that people intuitively know which generalisation is meant, using principles similar to Grice's conversational maxims (e.g. “Do not make your contribution more informative than is required.” [54]). As an example, he gives the triangles in Figure 2.3, which he claims suggest generalising to “All triangles”, “All right angled triangles” and “All right-angled isosceles triangles” respectively. The idea that Grice's maxims can (where relevant) be applied to diagrams – which I agree with – has implications for the design of diagram representations.

¹¹ Penrose's actual example involved 3D reasoning, which DIAMOND does not cover. However the nature of the reasoning is no different from that of the 2D proofs which DIAMOND does cover.



Figure 2.3. Triangles suggesting different generalisations (from left to right: all triangles, all right angled triangles and all right-angled isosceles triangles).

2.2 Representation schemes

2.2.1 Direct and indirect representations

Diagrammatic representations are by their nature quite specific, however the level of specificity varies. We introduce the term *direct* to informally describe the degree of this. A more direct diagram is one where the representation used is closely linked to its meaning. For example, drawing a triangle to reason about triangles. By contrast, an indirect diagram is one where the relation between sign and meaning is arbitrary, and based on convention. Constraint diagrams are an example of this, where a dot can represent anything from a spatial point to a person [11]. Textual representations¹² are always indirect.

In general, the closer the link between signifier and signified¹³, the more specific the representations are (i.e. more direct diagrams tend to be more specific). Specific representations lead to the generalisation problem described in §2.1.3. Also, it seems that the more specific the representations are, the harder it is to properly perform universal quantification. This is because there is extra information that the user must ignore. For example, it is easier to reason with ‘*let X represent any man...*’ than ‘*let the late Jon Barwise, who had fading brown hair and contributed so much to diagrammatic reasoning, represent any man...*’.

¹² We do not count as textual, representations such as ASCII art or rebuses (groups of letters, numbers or pictures that represent words or phrases) where the font and spatial arrangement of letters is important.

¹³ A *signifier* is the method (e.g. a word or picture) used to represent a concept (the *signified*); together they make up a *sign* [9].

Nevertheless, specific representations do seem to have strong advantages, as discussed in [10] and [49]. A strong link between the signifier and the signified allows people to mix semantic inference mechanisms such as model-checking with syntactic reasoning. This leads Chandrasekaran to characterise diagrammatic reasoning as a type of 'model instance based reasoning' [7]. Of particular relevance to this project, more direct diagrams give representations for geometric objects that are both very natural and seem to lend themselves well to diagrammatic reasoning.

Our domain is in geometry, hence we have adopted a system based on fairly direct diagrams. This gives us quite natural representations for many of the objects in the domain.

2.2.2 Diagrams and geometry

There is of course a long history of using diagrams in geometry – dating back further than reliable records [39]. Diagrams are routinely used in all areas of geometry, from abstract arrow chasing in topology, to sketches of concrete examples in mechanics.

Of particular relevance is the use of diagrams in teaching analysis. There have been a great many text books written on the subject of mathematical analysis, and it is not possible for us to survey even a fraction of them. No doubt most of them will use diagrams at some point, typically to illustrate concepts. This means that there are existing informal representation schemes which we can draw on, some of which are used on a wide enough basis as to constitute de-facto standards. We will use several established representation methods:

- Representing functions $f: \mathbb{R} \rightarrow \mathbb{R}$ by plotting a graph against horizontal and vertical axes.
- Representing functions between other spaces using arrows.
- Representing function application using arrows.
- In the high-school subject of *loci-shading*, shapes that do not include their borders are drawn using dotted lines. We will adapt this to represent *open sets*.

However, so far research on formal diagrammatic representations for geometry has focused on ruler-and-compass Euclidean geometry. We are unaware of formal diagrammatic representation schemes being developed for other aspects of geometry.

2.2.3 Representing quantifiers

One of the difficulties in extending diagrammatic reasoning beyond ruler-and-compass geometry is representing and reasoning about quantifiers. Since quantifiers are not related to any property of an object, or its relations to other objects, there is no obvious way in which to represent them. Most diagrammatic representation schemes do not cover quantifiers, and Schubert's scheme (c.f. §2.1.2) gives an example of how they can be problematic. One successful diagrammatic system with quantifiers is Howse *et al's* *constraint diagrams*, which are an extension of *spider diagrams* [11]. Constraint diagrams are an indirect representation for logical relations, and as such, they are very different in appearance to the logic we will develop here. Nevertheless, there are similarities at a structural level. A detailed comparison between our representation scheme and constraint diagrams is given in §7.2.

2.3 Existing diagrammatic reasoning systems

Most diagrammatic reasoning systems fall into two categories: those that use diagrams to guide an algebraic proof (which we call 'diagram guided theorem provers'), and those that convert diagrams into a high-level symbolic description and base inferences on this ('diagram interpretation theorem provers'). There are also a number of computer tools for visualising geometry problems (e.g. CINDERELLA [41]). However these are generally not designed for theorem proving.¹⁴

2.3.1 Diagram guided theorem provers

The Geometry Machine

Gelernter's Geometry Machine (GM) is an example of the first category¹⁵ [55]. It finds axiomatic proofs for Euclidean geometry theorems, using a diagram as a model to prune the search space. Branches in the search that are false in the diagram need not be further explored since they cannot lead to a proof. The GM was an early AI success, but not powerful enough to handle 'serious' problems [55]. However the basic idea of the GM

¹⁴ CINDERELLA does contain a theorem prover, but it is an algebraic one for checking user input. The user never sees the proofs.

¹⁵ Although it also performs a little diagram-interpretation, with some simple facts being read off the diagram.

appears, to a greater or lesser extent, in many other systems. In particular, Goldstein's Basic Theorem Prover is essentially an extension of the Geometry Machine [55].

“&”/Grover

An often cited modern system is “&”/GROVER, developed by Barker-Plummer, Bailin, & Ehrlichman [2]. This is a dual system of a diagram based proof planner (GROVER) and a standard theorem prover (the sequent calculus system “&”) which applies diagrammatic reasoning to set theory – a non-geometric domain. Instead of using the diagram as a model, it is used to generate proof plans. Figure 2.4 shows such a diagram. This reflects the authors' belief that diagrams carry 'meta-information' about a proof, such as the strategy or constructions to be used. Whilst this is a good idea in principle, GROVER relies too much on the user. The user must supply the diagram and convert it to a propositional description in order to input it – the program does not directly deal in diagrams. The user also has to verify the conversion of this description into logic statements. Moreover, GROVER diagrams are very specialised. They require the user to have expert knowledge of the system, plus a good understanding of the proof to be automated.

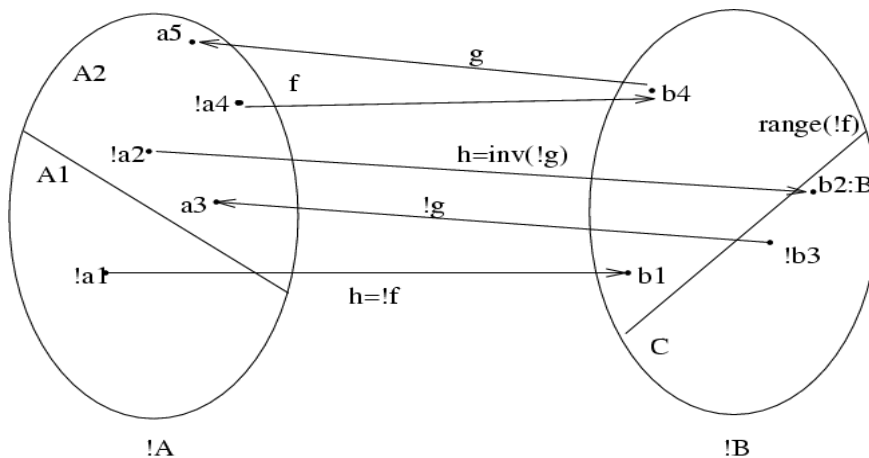


Figure 2.4. Diagram for proving the Schröder-Bernstein theorem¹⁶ with “&”/Grover

Summary

The Geometry Machine was very impressive for 1960. Gelernter's idea of using models to prune the search space has since been generalised to the algebraic method of 'semantic resolution' [55]. Model-checking provers based on diagrams do not seem to have moved on significantly from the Geometry Machine. Nor is it clear what purpose they now serve.

¹⁶ Schröder-Bernstein theorem: “If $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$ ” – but for A, B infinite.

Their output is an axiomatic algebraic proof; no more readable than comparable algebraic systems. The diagrams-as-proof-plans approach taken in “&”/GROVER also looks unpromising. It does not have the flexibility of proof planning systems, since a new diagram is required from the user for each theorem.

2.3.2 Diagram interpretation theorem provers

Diagram interpretation is a more promising category and has found some applications in graphical interfaces. However there is not a generally accepted way of interpreting diagrams [16], and different approaches result in different systems.

Visual languages

Perhaps the most cohesive body of work here is that done from a Natural Language Processing background. Various systems, such as PENGUINS, DIA GEN, and GEN ED have been developed [35]. They are generally designed for understanding and checking user generated diagrams. The envisioned application for such systems is not as theorem provers, but as intelligent front-ends [36].

HyperProof

Barwise & Etchemendy's HYPERPROOF currently sets the standard for educational applications of theorem provers [16]. It is aimed at philosophy students learning logic, and uses the blocksworld domain so that propositions can be given visual meaning. The system is multimodal, mixing diagrammatic and sentential representations and reasoning. Diagrammatic inferences in HYPERPROOF involve reading information from the diagram, or testing propositions against the diagram.

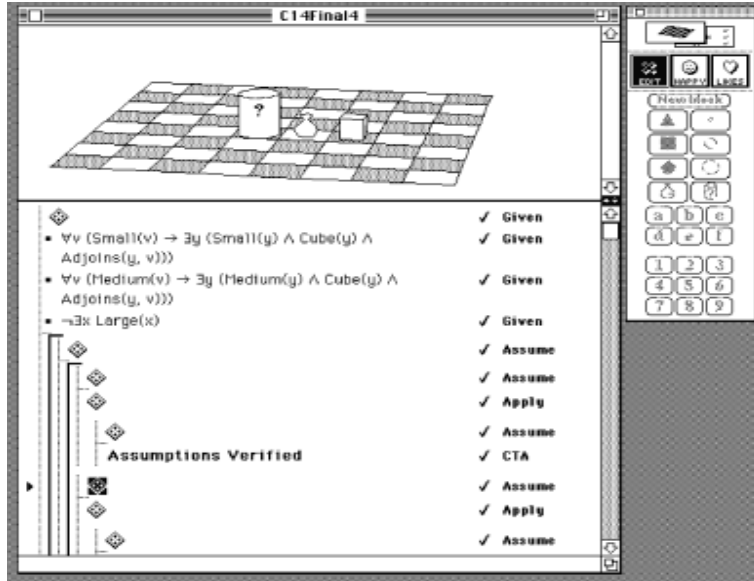


Figure 2.5. A HyperProof diagram

Fleuriot

In a promising crossover between geometric reasoning and geometric theorem proving, Fleuriot applied point-elimination methods from geometric theorem proving (extended to allow infinitesimal geometry) using the interactive system ISABELLE to prove the theorems of Newton's Principia in a geometrical manner [8]. This produced rigorous proofs from Newton's 'informal' ones.

Geometric theorem proving (GTP) involves using algebraic techniques (normally based on manipulating polynomials) to prove geometric theorems [55]. These techniques are very powerful – typically complete for universally-quantified conjectures in complex-valued geometries. Although these techniques can be based on geometric properties – as is the case with Fleuriot's work – GTP is quite different from diagrammatic reasoning, and the proofs are not particularly readable.

Fleuriot's work used non-standard analysis, which differs from ϵ - δ analysis in containing infinitesimal numbers (i.e. $\exists x$ such that $\forall n, 0 < x < 1/n$) and (non-equal) infinite numbers. It has been claimed that this is more intuitive [30]. My personal feeling is that non-standard analysis is more appealing in principle than in practice. Although non-standard analysis rehabilitates the (fairly) intuitive idea of infinitesimals, it introduces other difficulties (e.g., care is required with the different types of hyper-reals). Moreover, it is rarely taught.

Other systems

There are plenty of other systems that perform some form of diagram interpretation – including several formalisations of Venn diagram reasoning. Other approaches include Koedinger & Anderson's Diagram Configuration (DC) system. This was developed from a psychology/cognitive science perspective. The DC system is based on empirical data from observing the methods used by human experts to interpret geometry diagrams. It organises data into chunks representing key features from the diagram (e.g. triangles). This is processed by built in schemas (e.g. a right angled triangle schema), which can be thought of as inference rules. A back-chaining search for a proof is conducted by these schemas [24].

2.3.3 Dynamic reasoning systems

In all of the above systems, the diagram plays a static role. It is presented to the system as part of the initial input. The reasoning process uses a single diagram, and cannot modify any aspect of the diagram it is given. This approach matches – and is probably based on – the use of diagrams in text-books, and is quite different from algebraic (or, more generally, verbal) reasoning, which is a chain of logical steps forming an argument. However when people use diagrams in everyday situations, the diagram is often not static. The process of drawing the diagram can be important. It is not unknown for a diagram to become an unreadable mess by the end of the process, and yet still have served its purpose.

With algebra, the reasoning steps form a chain, but diagrams can be modified *in situ*.¹⁷ Sloman sees this as a cognitive advantage, reducing the load on working memory [50]. Under this view, the static diagrams in text-books are simply the final diagrams in the reasoning process. The end diagrams alone are given because they contain all the information, and it is easier to replay the reasoning on a single diagram than to flick through a comic book chain of evolving diagrams. Hence Sloman argues that “what [logic/verbal reasoning and visual/geometric transform reasoning] have in common is probably more important... than the differences”.

It is the dynamic use of diagrams that we will explore. There are only a very few systems at present that reason dynamically with diagrams. All of them use very different reasoning techniques, yet these techniques are not necessarily incompatible. We present an overview

¹⁷ There are some exceptions to this rule where algebra is modified in situ, such as cancelling terms in an equation by crossing them out.

of the main systems here, and a more detailed comparison with the system developed in this project in §8.

Diamond

Jamnik's DIAMOND system is one such system [25]. DIAMOND uses geometric reasoning about area to prove natural number arithmetic theorems. The user supplies proofs for example cases of a theorem, and from these examples a general proof is extracted and checked. The extraction process is symbolic, but the specific case proofs are done using entirely diagrammatic reasoning. Potentially it would be possible to automate the interactive element as a search with these geometric actions.

This project was initially conceived as extending Jamnik's work to a continuous domain. We view diagrammatic reasoning in a similar way, using example proofs for specific cases. However the differences between countable domain and continuous domain reasoning have led to a very different type of system. §7.1 gives a detailed comparison of this project and DIAMOND.

Hammer

Also in this category is a flawed logic developed by Hammer that mixes diagrams and sentences [14]¹⁸. Hammer's work builds on that of Shin in Venn diagrams. He extends this with extra notation and some inference rules to give a heterogeneous logic as expressive as first-order predicate logic. This was shown to be sound and complete. His work has several similarities to our own, including proofs formed from a sequence of diagrams, and the use of proof by contradiction. It is therefore of interest as it illustrates some pitfalls in formalising diagrammatic reasoning:

- 1) Hammer's definitions are slightly careless, and admit certain 'pathological' cases.¹⁹ This is very hard to guard against.
- 2) The meanings of the augmented diagrams are not particularly intuitive.
- 3) The inference rules are algebraic. That is, they are formulated and presented algebraically, and it is not clear that this could be done diagrammatically.

¹⁸ Hammer has since produced more appealing work ([15]), but it is not as closely related to the logic we develop here.

¹⁹ For example, Hammer's definitions speak of 'continuous closed curves', which allows space filling curves that do not have the properties he requires.

- 4) The inference rules cannot be directly applied (as formulated), but are instead used to verify that an inference step is correct. These steps are not intuitive, and without specific training many people might find them hard to understand or carry out.

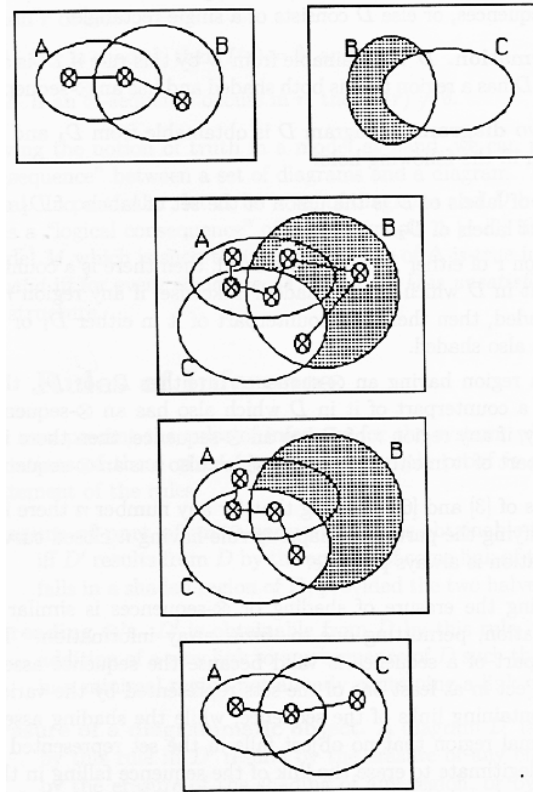


Figure 2.6. A proof in Hammer's diagrammatic logic of $x \in A \cup B$, $B \subset C \Rightarrow x \in A \cup C$.

The result is that it is probably harder to understand a proof given using his logic than a purely sentential one. To the best of my knowledge, it was never implemented.²⁰ It is thus hard to see what purpose Hammer's logic serves, except as an intellectual exercise, and as a warning that there are no automatic benefits to using diagrams. By extending diagram systems on purely logical criterion without considering ease-of-use issues, the final logic lost the very qualities that make diagrams attractive.

²⁰ As Ambrose Bierce's inventor said: "I have demonstrated the correctness of my details, the defects are merely basic and fundamental" [4].

Mondrian

Another system relevant to this project is Lieberman's MONDRIAN. This is not a theorem prover but a drawing program that can learn from examples [29]. Essentially, it is just a macro recorder, however it has similarities with the logic which we develop. These are:

- Drawing tools are presented by giving an example before/after transformation.
- General purpose tools are defined by working with specific examples.

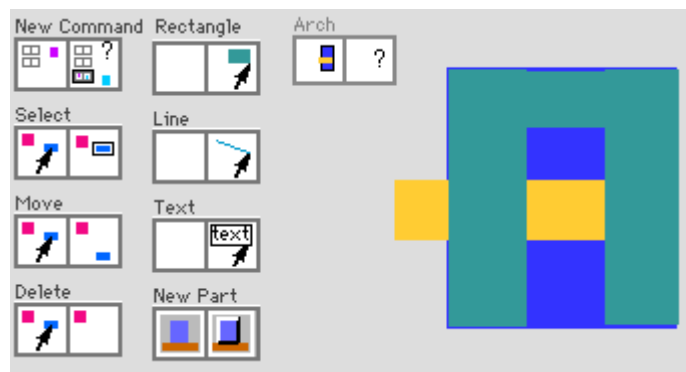


Figure 2.7. Screenshot of Mondrian. The 'dominoes' on the left are drawing tools, with drawing performed in the space to the right.

Other proposals

Meyer suggested coupling a visual language parser with a rewrite rule engine to create a diagram-based theorem prover for Euler Circles [35]. Such an approach is potentially more powerful than the redraw rules we consider (c.f. §4). The visual language framework would allow the rewrite rules to act at any level of abstraction. By contrast, the rules we consider can act at only one level, which is set by the diagram matching relation. Rewriting at an abstract level of description poses problems though for converting back into diagrams. A positive advantage of our approach is that the redraw rules are themselves diagrammatic. There is, thus, a natural, and hopefully intuitive, way of presenting them to the user. Visual language productions and arbitrary rewrite rules would not necessarily have clear diagrammatic interpretations.

2.4 Summary

Diagrams can be a powerful method of communication. However they are neither automatically nor universally so. Even a good diagrammatic technique may not be suitable

for all students, and a bad diagrammatic technique is less intuitive than algebraic methods. The focus should be on developing not just sound but intuitive reasoning techniques. Consideration of cognitive issues and frequent testing of representations and techniques on potential users should help with this.

Until recently, diagrams were considered unsuitable for rigorous mathematical proof, and this issue is still slightly controversial. Formalising diagrammatic reasoning presents problems beyond those of normal logic. However there is no reason why diagrammatic reasoning cannot be as rigorous as algebraic reasoning.

Diagrammatic reasoning has a long history in AI, dating back to Gelernter's Geometry Machine of 1960. After a period of neglect, the field has recently begun to attract serious attention again. However there are no dominant trends yet, and research continues in many directions. No work has been done in the domain that we wish to develop. Nor is there an existing system that appears suitable for adaptation.

3 Diagrammatic Reasoning: Problems & Challenges

This chapter continues and concludes the discussion on the problems involved in using diagrams in formal mathematics. It shows that developing a diagrammatic logic that is both powerful and sound is a daunting task. The challenges we identify here serve both to explain and motivate the hard work of the next chapter. In the previous chapter we looked at the question of whether diagrammatic reasoning can be formalised, and covered the most important and relevant related work. This included a discussion on generalisation in diagrammatic proofs. However there are other potential problems in using diagrams for theorem proving – mainly related to the need for *absolute* certainty – that were not covered. These include:

- Impossible drawings (i.e. drawings of non-existent objects)
- Optical illusions (i.e. drawings that trick the mind into seeing them inaccurately)
- Roughness of drawing – which prevents us from accurately drawing geometric objects
- Drawing mistakes
- Ambiguous drawing (i.e. drawings with more than one interpretation)

After examining these problems, we then show that sentential reasoning suffers from analogous potential pitfalls, albeit often in less severe forms. Hence whilst the problems discussed here are more acute for diagrammatic reasoning, they should not discourage us from developing diagrammatic logics.

We also look at why diagrammatic reasoning in analysis presents new challenges. These include representing disjunctions and negation, using contradiction and handling quantifiers – all of which require new solutions to those used in sentential reasoning.

Although this chapter makes general points where possible, the discussion here is both more technical and more specific to our reasoning method than in the previous chapter. In places

we will 'look ahead', outlining solutions which will be developed properly in later chapters. We focus on direct representations, where there is a strong link between diagrams and models, and primarily on 'geometry diagrams' where lines, points and curves in a diagram are used to represent lines, points and curves in the real plane \mathbb{R}^2 . There is a very natural mapping in geometry diagrams between domain objects and diagram objects. As we shall see though, this does not mean that creating a sound formal logic will be straightforward.

3.1 Problems in diagrammatic reasoning

3.1.1 Impossible drawings

Several diagrammatic reasoning methods are based around the idea that direct diagrams represent valid models (e.g. in Euclidean rule-and-compass reasoning, constructing a point shows that it exists). Unfortunately for the diagrams-models link, it is possible to draw impossible diagrams, that is, diagrams of non-existent objects (e.g. Figure 3.1²¹ and Figure 3.2²²). This can happen if the 'true' geometry of the drawing plane (i.e. the physical nature of the drawing plane) differs in some important way from the geometry we are reasoning about.

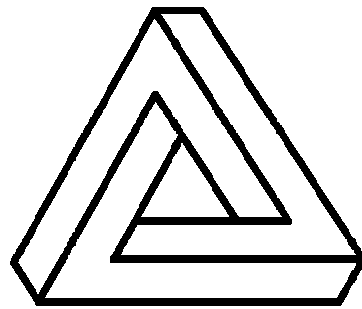


Figure 3.1. The physically impossible Penrose Triangle.

An example of this would be if we used Euclidean rule-and-compass reasoning, but performed on the surface of a sphere instead of a flat plane. It would be possible to construct cases that do not exist in Euclidean geometry, such as straight lines that meet at two separate points. The examples shown in Figure 3.1 and Figure 3.2 involve using diagonal lines to show depth as a way of representing 3D objects in 2D. The discrepancies between 3D space and this representation allow us to draw impossible objects.

21 Oscar Reutersvärd, 1934. Independently invented and popularised by Roger Penrose, 1958 [40].

22 Source unknown; based on designs by Oscar Reutersvärd, 1950s.

It is an assumption of this work that the drawing plane obeys Euclidean plane geometry sufficiently closely to avoid the construction of impossible objects. It is entirely plausible that the geometry of, say, a piece of paper or a computer screen is non-Euclidean at extreme scales. However our inference scheme will take into account the limits of drawing and measurement, so “sufficiently closely” will mean “up to detectable differences”. We examine this in more depth in §5.1.1.

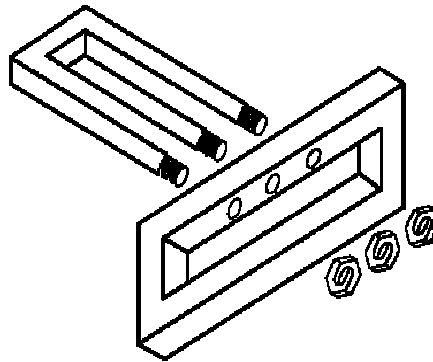


Figure 3.2. None of these objects are physically possible.

3.1.2 Optical illusions

Optical illusions play on idiosyncrasies of the mind's visual processing mechanisms to create false impressions. Figure 3.3 gives an example where parallel lines are made to appear slanted.²³ Other effects that can be created include misjudgements of size and seeing non-existent spots. The possibility of optical illusions calls into question the soundness of diagrammatic reasoning. Potentially a reasoning rule might be falsely applied because of such a misjudgement, or a false conclusion might be drawn from a diagrammatic proof.

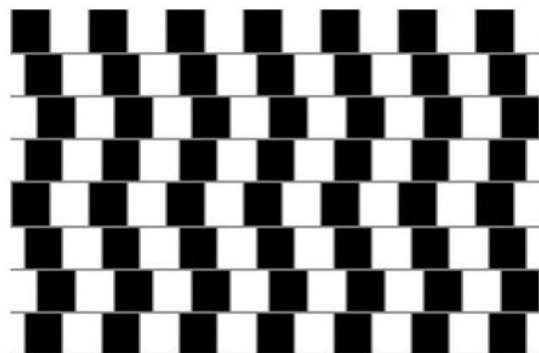


Figure 3.3. An optical illusion: the horizontal lines are parallel.

²³ Original source unknown.

Without a mature theory of optical illusions, it is not clear how serious this issue is. It seems reasonable to assume that optical illusions are unlikely to occur accidentally (and a malicious reasoner could use almost any reasoning scheme to mislead people). This would make them a theoretical concern rather than a practical worry. However we cannot completely disregard the possibility that some types of diagrammatic reasoning might naturally give rise to optical illusions. From the point of view of proving that our logic is sound, we note that computer implementations do not suffer from optical illusions.²⁴ So whilst a user might conceivably be deceived as to what was being proved, they would be unable to prove anything false. Eventually they would probably discover their mistake, and a careful examination of the diagrams would always dispel the illusion.

3.1.3 Roughness of drawing

We can only draw to a certain level of accuracy. Also, we can only measure and test a drawing to a certain level of accuracy. So all drawings have some 'roughness' to them. This gives rise to several problems: a potential vagueness in the representation, restrictions on what can be represented and the potential to hide inaccurate representations.

Vagueness of representation

The problem is this: we wish to reason about geometry, where lines have no width and points have no size. However if we draw a line, it will have width, and a point will cover a certain area. There is also a limit on how precisely we can position or measure objects drawn in a diagram. These limitations mean that an object in a diagram might represent many different objects in the domain. Moreover, these objects might be different in important ways. For example, if we draw sets by shading points within the set, then the real line \mathbb{R} and the rationals \mathbb{Q} will appear identical (see Figure 3.4). It is therefore crucial that our diagram logic must have a certain tolerance towards rough drawing.

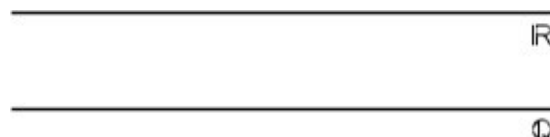
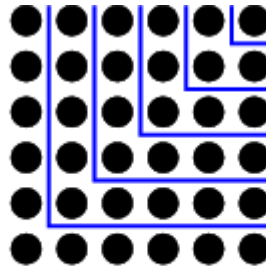


Figure 3.4. An attempt to plot \mathbb{R} and \mathbb{Q} with points plotted as dots of radius 1 printing pt.

²⁴ That is, a computer implementation with access to a symbolic description of the diagrams. Computer vision systems that use heuristics to parse images can be fooled by optical illusions, but that is not relevant here.

Let us call the geometric objects we want to represent an *idealised* diagram, and our actual physical representation a *physical* diagram. Once scaling, orientation and drawing tools have been fixed, there is a simple mapping from idealised diagrams to physical diagrams. However, roughness of drawing means that this mapping is many-to-one: each physical diagram could have been created by many different idealised diagrams. Another reasoner might think – quite validly – that it was one of these other idealised diagrams which was intended.

Note that this problem does not occur in all domains. For example, Jamnik's DIAMOND system works in the discrete domain of natural number theory. It uses what Jamnik calls a 'topological representation' where roughness of drawing is not a problem [24]. Figure 3.5 shows an example of this representation method. The implicit grid on which drawing is done means that small errors in drawing will not affect the diagram's interpretation. However this problem cannot be avoided in continuous domains²⁵.



$$1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

Figure 3.5. An example of Jamnik's 'topological representation'.

One solution is to discretise the domain, that is, to approximate it with a discrete grid. This is not immediately suitable for analysis though, because infinitesimal differences are often very important. For example, the difference between the open set $(0,1)$ and the closed set $[0,1]$ is just two infinitesimally small points. Any discretisation would lose this crucial difference.

²⁵ It will also occur in dense domains, e.g. domains that quantify over the rationals.

However any physical drawing already has an implicit discretisation.²⁶ To see this, consider the related problem of representing sound (a continuous domain). The modern method used is to digitise it, converting a continuous domain into a discrete one. If the sampling is fine enough, people cannot tell the difference. Similarly, a flat picture can be captured as a 'bitmap image', and if the resolution is fine enough, we cannot tell the difference. The key point is that human senses can only detect differences up to a certain level. And if two representations are indistinguishable, they must be treated as identical by any logic that handles them.

In principle, though it might sound contradictory, it is possible to avoid this implicit discretisation by using computers instead of pen and paper. This is because a computer can allow us to view a diagram at arbitrary resolutions. If the internal representation from which each view is created does not perform any rounding, then we have not digitised. By zooming in enough, it will be possible to tell the difference between even minutely different objects. However the user cannot know how many times they must zoom in to check a given property. For example, if two points in a plane are not equal, this can always be detected by plotting them at a high enough resolution, but we cannot tell in advance what level of resolution will be necessary, so plotting the points will never tell us that they are equal. Moreover some objects – such as $(0,1)$ and $[0,1]$ – will be indistinguishable at *any* level of magnification.

We conclude that vagueness of representation – that is, the many-to-one mapping between idealised and physical diagrams – is an unavoidable problem when reasoning in a continuous domain.

We will thus accept that our drawings (physical diagrams) will be rough approximations of the objects we wish to represent (idealised diagrams). Each object in a physical diagram can represent a class of domain objects. For example, if we have a point that appears to be at $(1,1)$, then this might actually represent any point within ϵ of $(1,1)$, where ϵ depends on drawing scale, drawing tools (i.e. dot size), and the limits of human vision. Thus there will be a one-to-many mapping from physical diagrams to ideal diagrams.

²⁶ We assume here, and throughout this thesis, that space is continuous, and a line drawing on a piece of paper is a continuous representation. Quantum mechanics sheds some doubt on this assumption. However since we move on to say that our representations will be discrete, this assumption is not important.

Limitations of what can be drawn

In the discussion above, we accepted that roughness of drawing prevents us from representing any object precisely. It also bars us from directly drawing some objects at all (e.g. \mathbb{Q}) as their appearance would inevitably be misleading (c.f. Figure 3.4). This places restrictions on what we can reason about without using indirect annotations. For example, we cannot directly represent points that are too close to be distinguishable, curves which look like straight lines, functions with small (i.e. undetectable when drawn) discontinuities, etc. These are reasonably strong limitations for our domain.

Our solution to this is to use direct representations where possible, with indirect annotation where necessary. By adding annotation, we can make vague aspects of a diagram more precise. Thereby we can regain accuracy and extend what can be represented – although at the cost of a more complex representation scheme. Our diagrams will mix object representations and explicit statements of relations. Explicit statements will have 'priority' over diagrammatic inferences: where an explicit statement contradicts the appearance of the diagram objects, the explicit statement is taken to be true and the diagrammatic inference is ignored. For example, suppose two points appear identical, but the diagram has annotation stating that they are separate, then they are interpreted as being separate.

Lost details are 'important' if without them a desired relation cannot be legitimately inferred, or an undesired relation can be legitimately inferred. In these cases, notation must be added either to assert the desired relation or negate the undesired one.

Most of the lost details, such as the precise position of a point, will not be important. If we do want to precisely specify a point, we can do so by annotating it. For example, if we draw a point that appears to be at $(1,1)$ *and we label the point $(1,1)$* , then it can only represent the point $(1,1)$; the vagueness introduced by roughness of drawing is removed by extra notation. We anticipate that most such annotation will be reasonably simple statements. However diagram annotations can contain algebraic statements and be as complex as required.

Hidden inaccuracy

The problem: it is possible to draw diagrams which use roughness of drawing to create false impressions. Figure 3.6 is a classic example²⁷ of this. It shows 4 blocks being rearranged to form the same triangle – but with one less square. The flaw is that the two small triangles are not similar triangles, and therefore the large composite triangles are not really triangles at all – because their hypotenuses are not straight lines – but two different quadrilaterals. The illusion is quite persuasive, since it creates a big discrepancy by magnifying a small inaccuracy.

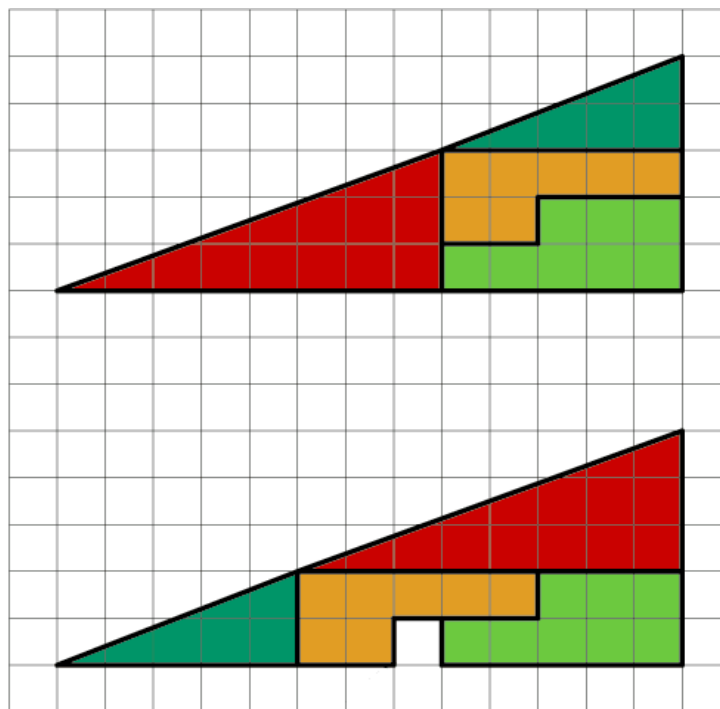


Figure 3.6. Rearranging the 4 shaded blocks seems to change their total area.

To solve this, we check when drawing objects that their appearance matches with what is known about them. If there are any discrepancies, then the diagram is annotated to show these. Hence Figure 3.6 would not be allowed unless the hypotenuse 'lines' were labelled to indicate that they aren't really lines, which would ruin the illusion.

Clarity

Another issue related to roughness of drawing is the problem of unclear diagrams. It is possible for diagrams to be too cluttered to be correctly read. This occurs when objects (or

²⁷ Original source unknown.

relational statements) are drawn too small or too close together to be properly identified. Potentially this issue could be dealt with by developing a theory of clear representations. Unfortunately, that goes beyond the scope of this project. For our purposes, we must assume that users will not create unclear diagrams.

3.1.4 Drawing mistakes

One advantage of using computers is that lines, angles, curves, etc. can all be plotted accurately. This eliminates the possibility of accidentally constructing inaccurate diagrams. However drawing mistakes could still arise in several ways:

- 1) Rounding errors can accumulate to produce a gross inaccuracy.
- 2) We may not know how to draw an object. That is, we cannot reliably calculate how it should be drawn. For example, given a set X and a function f , reliably calculating how to draw $f(X)$ may be non-trivial.
- 3) We may not be able to evaluate a condition, and therefore will not know if our diagram is or isn't accurate. Conditions such as " $x < y$ " are easy to evaluate. However conditions such as " X is an open set" cannot, in general, be evaluated.

A sound logic must either prevent inaccuracy from occurring, or work regardless of it.

In most of the reasoning we will consider, inaccurate diagrams are not actually a problem. This is because our logic prevents anything being inferred from the diagram that could not be proved algebraically. Where inaccurate diagrams do become important in our work is in counter-example reasoning (i.e. existence theorems), where we use the link between diagrams and models to simplify the proof process (using the approach: "if we can draw it, it exists"). For this to work, we require that our drawing methods are accurate, which places limitations on where such reasoning can be used.

3.1.5 Ambiguity

By ambiguity we mean that one diagram may be interpreted in multiple ways. This can arise from roughness of the representation, as discussed in §3.1.3. However ambiguity can also arise even when all the parts of a diagram are drawn clearly. Figure 3.7 gives a picturesque example of this;²⁸ Figure 3.8 gives a more prosaic example of greater relevance to the reasoning we consider. This is not a trivial issue, especially in more complex diagrams. For a formal logic, it can be eliminated by placing restrictions on the drawing. If only a limited

²⁸ Original source unknown.

range of objects are possible, and both their individual and compositional interpretations are fixed, then the diagrams will not be ambiguous within the context of the logic.



Figure 3.7. Young or old? (hint: the young woman's chin is the old woman's nose).

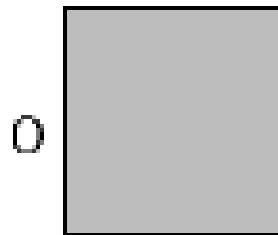


Figure 3.8. Is the 'o' a label for the square, a label for just one side, or is it a separate circle?

3.2 Algebraic pitfalls

The pitfalls of diagrammatic reasoning, such as those described above, can be presented as an argument against the validity of diagrammatic reasoning. Critics of diagrammatic reasoning tend to compare the naive *informal* use of diagrams with the formalised use of algebra. It is therefore worth noting that informal algebraic reasoning (which – outside of certain specialised communities – is more commonly used than formal logic) is also prone to error.

Examples abound, even at the highest level (for example, the mistakes in Wiles' first proof of Fermat's Last Theorem [47]. It is altogether possible that these would not have been detected in a less high profile proof). The domain we consider is particularly prone to error, and even the great Cauchy made mistakes [32][28]. Below we present a selection of errors in sentential reasoning roughly analogous to those listed above for diagrammatic reasoning. Of course none of these errors invalidates algebraic reasoning as a form of mathematical proof. Formal algebraic logic is immune to all of them. The same can be true for diagrammatic reasoning.

3.2.1 Roughness of representation

Rounding errors are an algebraic equivalent to roughness of drawing in diagrammatic representations. In Figure 3.6, we showed how magnifying a small drawing inaccuracy could create a big error. A real-life sentential equivalent of this trick is 'salami attack' computer fraud:

“[There is] an automated form of computer abuse called the salami attack, which works on financial data. This technique causes small amounts of assets to be removed from a larger pool. The stolen assets are removed one slice at a time (hence the name salami). Usually, the amount stolen each time is so small that the victim of the salami fraud never even notices.

One theoretical financial salami attack (it's assumed the status of an urban accounting legend and has never actually been known to have been attempted) involves rounding off balances, crediting the rounded off amount to a specific account. Suppose that savings accounts in a bank earn 2.3%. Obviously, not all of the computations result in two-place decimals. In most cases, the new balance, after the interest is added, extends out to three, four, or five decimals. What happens to the remainders? Consider a bank account containing \$22,500 at the beginning of the year. A year's worth of interest at 2.3% is \$517.50, but after the first month the accumulated interest is \$43.125. Is the customer credited with \$43.12 or \$43.13? Would most customers notice the difference? What if someone were funneling off this extra tenth of a penny from thousands of accounts every month? Although this particular salami hasn't to our knowledge been attempted, salamis that shave a quarter on up have been tried.

A clever thief can use [a salami program] that puts all of the rounded off values into his account. A tiny percentage of pennies may not sound like much until you add up thousands of accounts, month after month. Criminals using this scheme have been able to steal many thousands of dollars. They are sometimes discovered by a bank audit. More often, they are detected only when they use their new-found gains to entertain a life style that is not supported by their legitimate income.”[23]

3.2.2 Limitations of what can be represented

As noted in §3.1.3, we cannot diagrammatically represent everything in a continuous domain. It is worth noting that algebraic reasoning suffers from the same drawback: There are objects in real analysis which simply cannot be represented. Indeed, technically *almost all* objects cannot be accurately represented²⁹. That said, the problem is worse for diagrams, as there are interesting geometric objects, such as \mathbb{Q} , which can be represented algebraically, but cannot be drawn without causing confusion.

3.2.3 Textual illusions

How many Fs are there in the following sentence?

"FINISHED FILES ARE THE RESULT OF SCIENTIFIC STUDY
COMBINED WITH THE EXPERIENCE OF YEARS."

(Most people find 3, but in fact there are 5)

Figure 3.9. A textual illusion.

It might seem that the problem of optical illusions is unique to visual representations. However sentential equivalents are possible, albeit much rarer. The puzzle in Figure 3.9 gives one example where two letters 'vanish' [56]. The illusion would not be serious, except that formal logic relies on the precise manipulation of symbols in sentences.

3.2.4 Ambiguity

Statements with multiple interpretations are a familiar problem in linguistics. Lexical ambiguity occurs when a simple expression, like 'bank', has more than one meaning. Structural ambiguity occurs when a complex expression has more than one meaning, but not because any of its parts are lexically ambiguous. For example: "I once shot an elephant in my pyjamas. What it was doing in my pyjamas, I'll never know." - Groucho Marx³⁰. Formal mathematics simply outlaws lexical ambiguity and prevents structural ambiguity by fixing operator precedence.

²⁹ The proof of this is simple: the domain is uncountable, whilst algebraic representations can at best distinguish between a countable number of objects. Therefore there exist an uncountable number of objects without specific representations, greatly outnumbering those objects with representations.

³⁰ "Animal Crackers" Paramount Pictures, 1930.

3.2.5 Hidden assumptions

Hidden assumptions crop up in lots of places when reasoning algebraically. The classic example is dividing by zero. We present an example of this in Figure 3.10.³¹ However there are many other pitfalls: taking the limit of a non-convergent sequence, re-arranging terms in a sequence that is convergent but not uniformly convergent, differentiating a function that has partial derivatives but is not actually differentiable, etc., etc.

Let $a=b$
Then $a^2 = ab$
 $a^2 + a^2 = a^2 + ab$
 $2a^2 = a^2 + ab$
 $2a^2 - 2ab = a^2 + ab - 2ab$
and $2a^2 - 2ab = a^2 - ab$
This can be written as $2(a^2 - ab) = 1(a^2 - ab)$
Cancelling the $(a^2 - ab)$ from both sides gives
 $1=2$

Figure 3.10. A fallacious proof that $1=2$.

3.3 Challenges in diagram logic

In this section we look at various desirable features of conventional logic, which are not so easy to achieve in diagrammatic logic. Many of the problems in reproducing these features in diagrammatic reasoning stem from the close link between diagrams and models. The solutions we adopt to these problems are presented in chapter 4.

3.3.1 Disjunction

Pictures do not naturally represent disjunctions. Consider the statement “ p or q ”. In any given model of “ p or q ”, either p is false, q is false or “ p and q ” is true. A logic that includes disjunction must allow for all of these possibilities. Hence if we equate diagrams with models, then a diagram cannot represent “ p or q ”, because it will represent only one of the three possible cases.

³¹ Original source unknown.

There are at least two ways round this. Our solution will be to use multiple diagrams (c.f. §4.2.6). The other alternative, as implemented in spider diagrams, is to introduce a special notation for disjunction [11]. Spiders are good for the case $x \in A \cup B$ (since they avoid splitting into multiple diagrams). However they only allow one type of relation – membership – to be represented. Constraint diagrams generalise spider diagrams to represent other relations, but the disjunction notation has not been generalised, and it is not clear that it could be without producing unintuitive diagrams. HYPERPROOF also uses special disjunction notation, with a shape that represents 'shape unknown', and a '?' sign for size unknown. Again, this is not a general solution, since (a) it only handles two relations, and (b) it cannot represent cases with partial knowledge, such as “ X is a cube *or* a cylinder”.

3.3.2 Negation

Often relation statements can be expressed by drawing a diagram that exhibits the desired relation. Negative relation statements are more problematic. Arguably for some relations the same approach – of drawing a suitable example – still works (e.g. $\text{not}(x \in X)$ has an intuitive diagrammatic representation). In general though, we cannot represent negative statements by drawing an example. For example, $\text{black}(\text{raven})$ is easy to represent diagrammatically by drawing a black raven, but the relation $\text{not}(\text{black}(\text{apple}))$ is much harder (would one draw an apple in every shade but black?). Representing relations by examples is only feasible when there are a small number of cases. For negative relations, there can easily be a large if not infinite number of cases (e.g. $\text{not}(x=7)$).

3.3.3 Contradiction

If we equate diagrams with models, then diagrams are necessarily consistent. This would take away the powerful tool of proof by contradiction, and limit diagrammatic reasoning to constructive logics.

3.3.4 Extending to new concepts and domains

Algebra is quite easily extended to new domains. When faced with new concepts, we can simply define new symbols and terms to cover them.³² The same cannot be said for diagrams.

³² Russell does not consider this to be a simple process: “A good notation has a subtlety and suggestiveness which at times make it almost seem like a live teacher.”[63] However he didn't consider diagrammatic notation, and it is certainly true that extending algebraic notation is easy by comparison.

The strength of diagrammatic representations is that they are meaningful themselves (e.g. using *inside* for subset implies that the relation is transitive, using dotted lines for an open set suggests that the border isn't included, etc.). However this also makes them hard to extend to new concepts. A new concept should have a representation that:

- 1) Conveys information about the concept.
- 2) Fits in with the existing representations (e.g. if you use colour to represent quantifier type, you should not use colour to represent anything else).
- 3) Is not misleading.

This makes extending a diagrammatic language to handle new concepts potentially much harder than for algebra. The converse of this is that successful diagrammatic representations are powerful aids to reasoning. An important part of this project has been finding a suitable representation scheme for the domain.

3.3.5 Quantifiers

The lack of an obvious representation for quantifiers has already been discussed in §2.2.3. Here we note two more problems relating to their use.

Quantifier hierarchy

Our chosen domain contains concepts defined using alternating quantifiers. Quantifier hierarchy – that is, the order in which the quantifiers appear – makes an important difference here. Thus any representation we use for quantifiers must also capture their hierarchy. In sentential reasoning, quantifier hierarchy is determined by reading from left-to-right. However, the use of two dimensions with several spatial relations being significant removes the neat left-to-right ordering on objects that we have in sentential reasoning. So representing quantifier hierarchy in diagrams requires a new solution.

Existential import

It is natural when presented with a diagram that shows various objects to assume that at least one example of the facts depicted does exist. Indeed, the link between diagrams and models probably means that the diagram is an example of the facts it depicts. The natural way of interpreting diagrams – that an example of the facts depicted does exist – means that the universal quantifier has *existential import*. That is, the statement $\forall x \in X$ implies $\exists x \in X$. This is also true in aristotelian logic ([26]) and natural language, but of course false in predicate calculus. It becomes problematic when we wish to make statements about universally

quantified objects, which might not exist. In such cases there is a potentially dangerous gap between the natural reading of a diagram and conventional mathematics. We discuss this further in §4.2.7.

3.4 Summary

In this chapter we have examined some of the ways in which diagrammatic reasoning could lead to mistakes, and also the ways in which diagrammatic reasoning is harder than algebraic reasoning. We conclude that there are serious challenges involved in developing a formal diagram logic. These include:

- Working with roughly drawn imprecise diagrams.
- The possibility of drawing mistakes.
- Ambiguous drawing.
- Representing disjunctions, contradictions and negation.
- Reasoning with quantifiers.
- Generalising a proof correctly.

This list might suggest that diagrammatic theorem proving is not worth pursuing. However we have outlined solutions to some of these problems (typically based on accepting the limitations of diagrams, and working either with or around them), and also shown that analogous problems occur in sentential reasoning. As we shall demonstrate in chapters 4 and 5, these problems do not prevent sound diagram logics from being possible, even for difficult domains.

4 A Diagram Logic for Analysis

Proofs

This chapter sets out the methods used for representation and reasoning, and explains how they work. We start by presenting an example demonstrating the kind of reasoning we wish to formalise. This will be quite informal, as its purpose is to illustrate the differences from standard reasoning in the domain, and to show some of the challenges involved. §4.2 discusses how our logic – which we call *dynamic diagram logic* (or DDL) – works. We summarise its conclusions here:

- 1) The diagrams mix graphical and sentential elements, but with a preference for graphical representations when feasible. The representations used are designed to give clear diagrams and make the reasoning as natural as possible.
- 2) Both the inference rules and the proofs are made from these diagrams, rather than using diagrams to guide an algebraic proof.
- 3) The reasoning is 'dynamic' reasoning, where the process of drawing diagrams is important (as opposed to 'static' reasoning, which interprets a finished input diagram).
- 4) The reasoning involves a mix of diagrammatic reasoning rules, plus implicit inferences based on the diagrammatic representations.
- 5) The reasoning uses specific examples from the domain rather than universally quantified objects. Where possible, it uses examples that can actually be drawn in two dimensions. This way the example object can represent itself, rather than being represented as an abstract label. Theorems with a continuum of cases are proved by considering one specific (but generic) case.
- 6) The correct generalisation is extracted from the structure of the proof, rather than being part of the statement of the theorem.

§4.3 then defines the formal framework for DDL. This section is crucial to showing the soundness of the logic, but not necessary for using the logic. In a sense, the first three

sections cover the same ground, first illustrating (informally), then discussing issues involved in formalising, then formalising.

The fourth section then examines how DDL can be used to reason about analysis – giving a system we call *DDLA*. It covers the representation schemes we use, the 'built-in' inference rules, and some example axioms. DDLA follows ϵ - δ analysis as closely as possible. This should help in developing proofs, and also ensure that the logic developed will be useful for teaching the subject. The logic set out in this chapter will then be analysed for soundness in chapter 5.

4.1 An example proof: $f(x) = 1/x$ is continuous on $(0, \infty)$

The proof of this statement is reasonably complex, illustrating most aspects of our logic. The proof will invent representation schemes as it goes, which are discussed later in §4.4.

We prove f is continuous by showing that f is surjective (Figure 4.1), f is decreasing (Figure 4.3) and that surjective and decreasing implies continuous (Figure 4.5). The algebraic proof follows the same plan. We assume for now that f is well defined; this will be proved later, both algebraically and diagrammatically.

4.1.1 Diagrammatic proof of the theorem

f is surjective

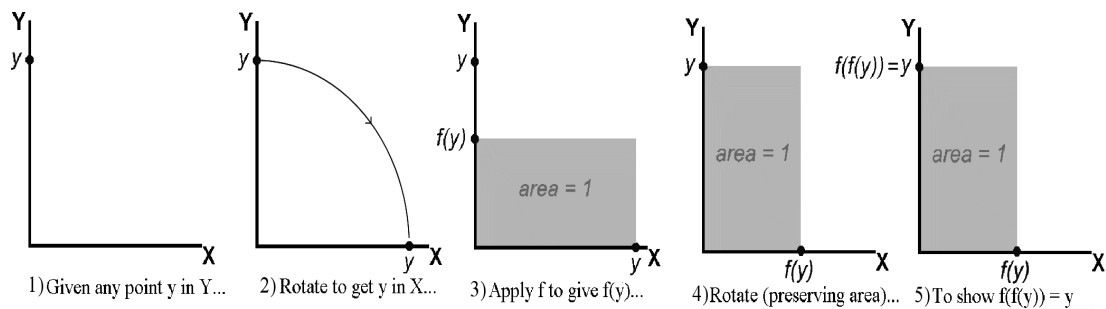


Figure 4.1. Diagrammatic proof that $f(x)$ is a surjective function (i.e. $\forall y \exists x. f(x) = y$)

Figure 4.1 shows a diagrammatic proof, which should be read from left to right as a sequence of reasoning steps. We first consider an arbitrary point $y \in Y$ (diagram 1), then get $y \in X$, since $Y=X=(0,\infty)$ (diagram 2). We then generate a point $f(y) \in Y$ and use the fact that $\forall x, x.f(x)=1$ (from the definition of f) to create a rectangle of area 1 (diagram 3). We can now rotate this rectangle (preserving its area) to get a rectangle of area 1 with sides $y \in Y, f(y) \in X$ (diagram 4). This implies that $f(f(y))=y$ so we have found a pre-image for y as required (diagram 5).

This property – $x.f(x)=1$ – will be our diagrammatic definition for f , which we codify using two rules. These are shown in Figure 4.2. Rules are defined by antecedent and consequent diagrams, with the antecedent drawn above the consequent. The left-hand rule states that given points $x, f(x)$, the area of the rectangle they generate will be 1. The right-hand rule is the converse; if two points x, y generate a rectangle of area 1, then $y=f(x)$. The change here is much more subtle: only a label – $f(x)$ – is added.³³

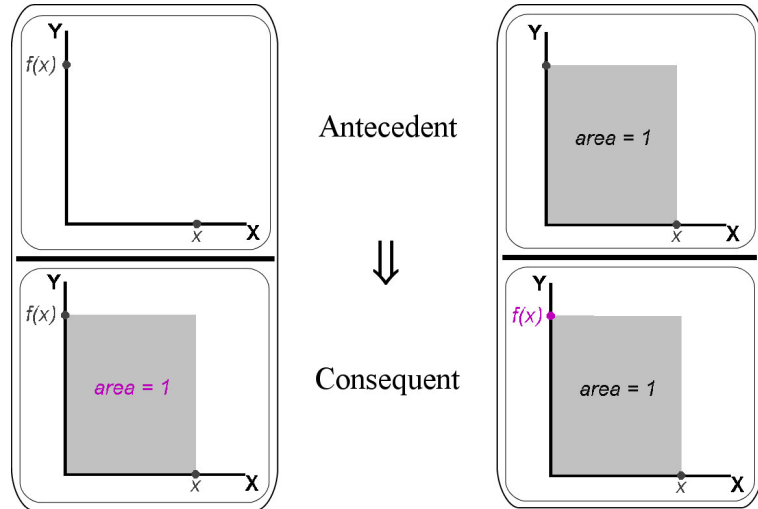


Figure 4.2. Diagrammatic definition for $f(x) = 1/x$.

Note that when defining a function we cannot simply use the graph of that function, as this would not be rigorous without tackling questions of what resolution it was drawn at. Depending on the resolution of the graph, there may be hidden features such as spikes and

³³ Because changes can be subtle, we sometimes use a highlighting technique to emphasise them. This is implemented in our **Dr.Doodle** system (c.f. §6.3), although not required by the specification.

jumps.³⁴ Sketching the graph in might serve as a useful prompt to reasoning, but to give rigorous proofs we need a more concrete property, such as $x \cdot 1/x = 1$. Such properties should also have a strong visual aspect if they are to produce good diagrammatic proofs.

f is decreasing

The next stage of the proof (Figure 4.12) involves performing a case split, and showing that only one of the cases is valid. The other cases are eliminated by deriving a contradiction – this is done by creating two rectangles that should have the same area, but clearly do not. We start by drawing two points x_1, x_2 such that $x_1 < x_2$. We then plot $f(x_1), f(x_2)$ and consider the three possible orderings of these points. By using the fact that $f(x) \cdot x = 1$ (the left-hand rule from Figure 4.2), we show that two of these orderings are impossible. This leaves the third option – that $x_1 < x_2 \Rightarrow f(x_1) > f(x_2)$, i.e. the function is decreasing.

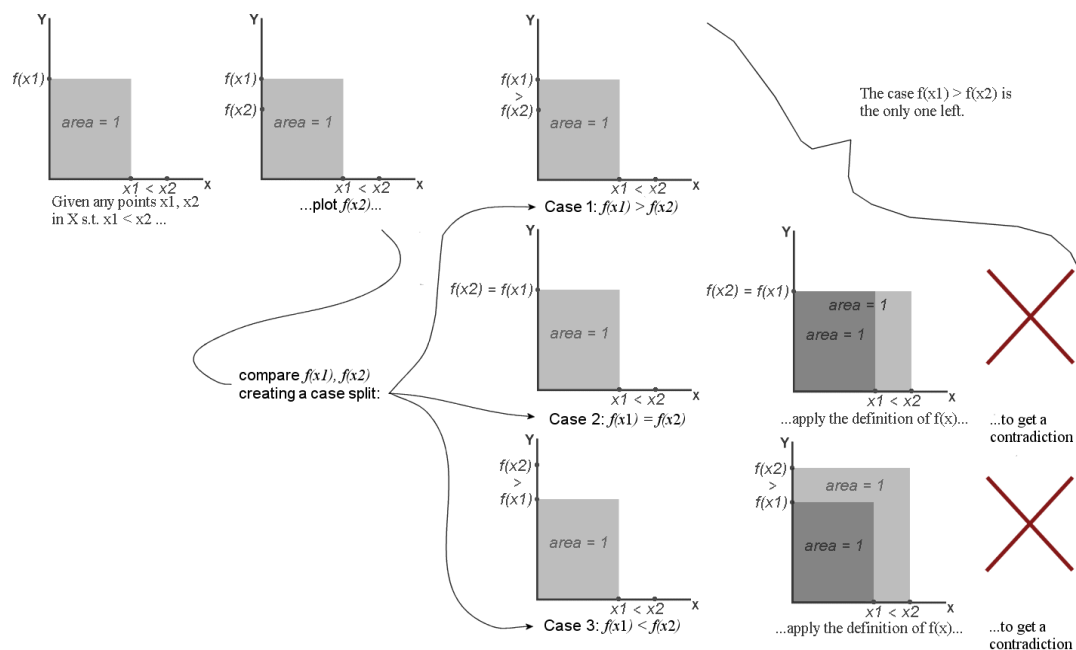


Figure 4.3. Diagrammatic proof that $f(x)=1/x$ is a decreasing function.

Surjective and decreasing \Rightarrow continuous

The last stage of the proof is the most complicated. Let us first consider what it means to be continuous. Continuous functions map nearby points to nearby points. In ϵ - δ analysis, the concept of 'nearby' is formalised as 'within a ball of arbitrarily small radius' (traditionally

³⁴ We *could* construct a logic where reasoning directly from the graph is rigorous (e.g. by restricting to uniformly continuous functions, which we can plot with guaranteed error bounds.). However we will not explore that option here, as we want to consider all functions – even the pathological ones.

this radius is denoted by ϵ). It gives rise to the following definition for continuity: “If a function is continuous, then given any point x and any $\epsilon > 0$, there exists a $\delta > 0$ such that $f(B_\delta(x)) \subset B_\epsilon(f(x))$ ” (where $B_r(x)$ = the open ball of radius r about point x , and we define $f: \text{sets} \rightarrow \text{sets}$ from $f: \mathbb{R} \rightarrow \mathbb{R}$ in the normal manner). We represent this diagrammatically as shown in Figure 4.4.

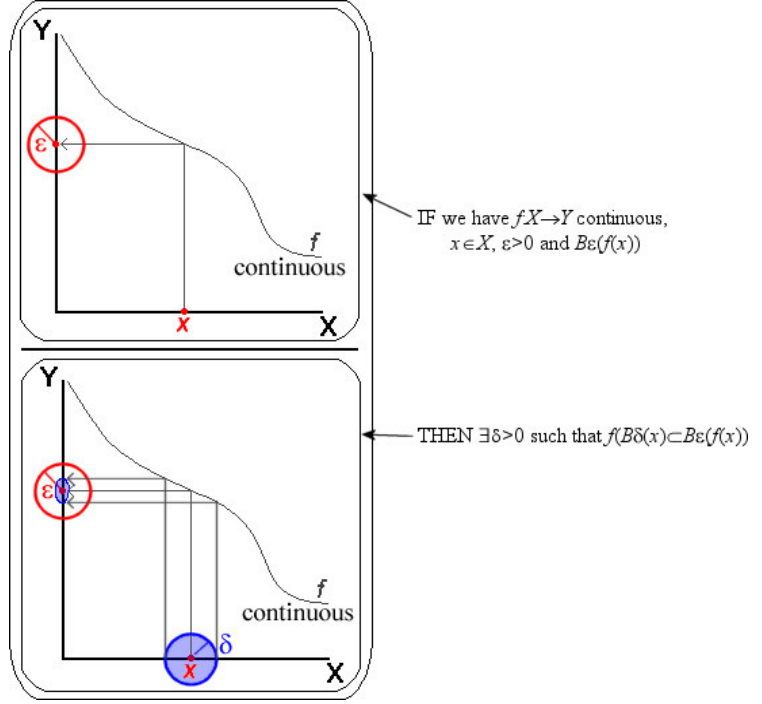


Figure 4.4. Redraw rule defining continuity.

However for this proof, we wish to show that f is continuous, which means using the converse rule:

Given $f: X \rightarrow Y$, if $\forall x \in X, \forall \epsilon > 0, \exists \delta > 0$ such that $f(B_\delta(x)) \subset B_\epsilon(f(x))$ then f is continuous

Representing this diagrammatically poses certain problems, which we discuss in §4.2.7. Our solution will be to use an animated antecedent, which allows us to handle the alternating quantifiers in the definition above. For now, we observe that to prove f is continuous, we must demonstrate that the rule shown in Figure 4.4 holds (i.e. given an arbitrary x and ϵ , we must find a suitable δ).

Figure 4.5 now presents our proof of the final lemma. Diagrams 2 and 3 create an arbitrary x and ϵ . Because nothing is known about x and ϵ , they behave like universally quantified

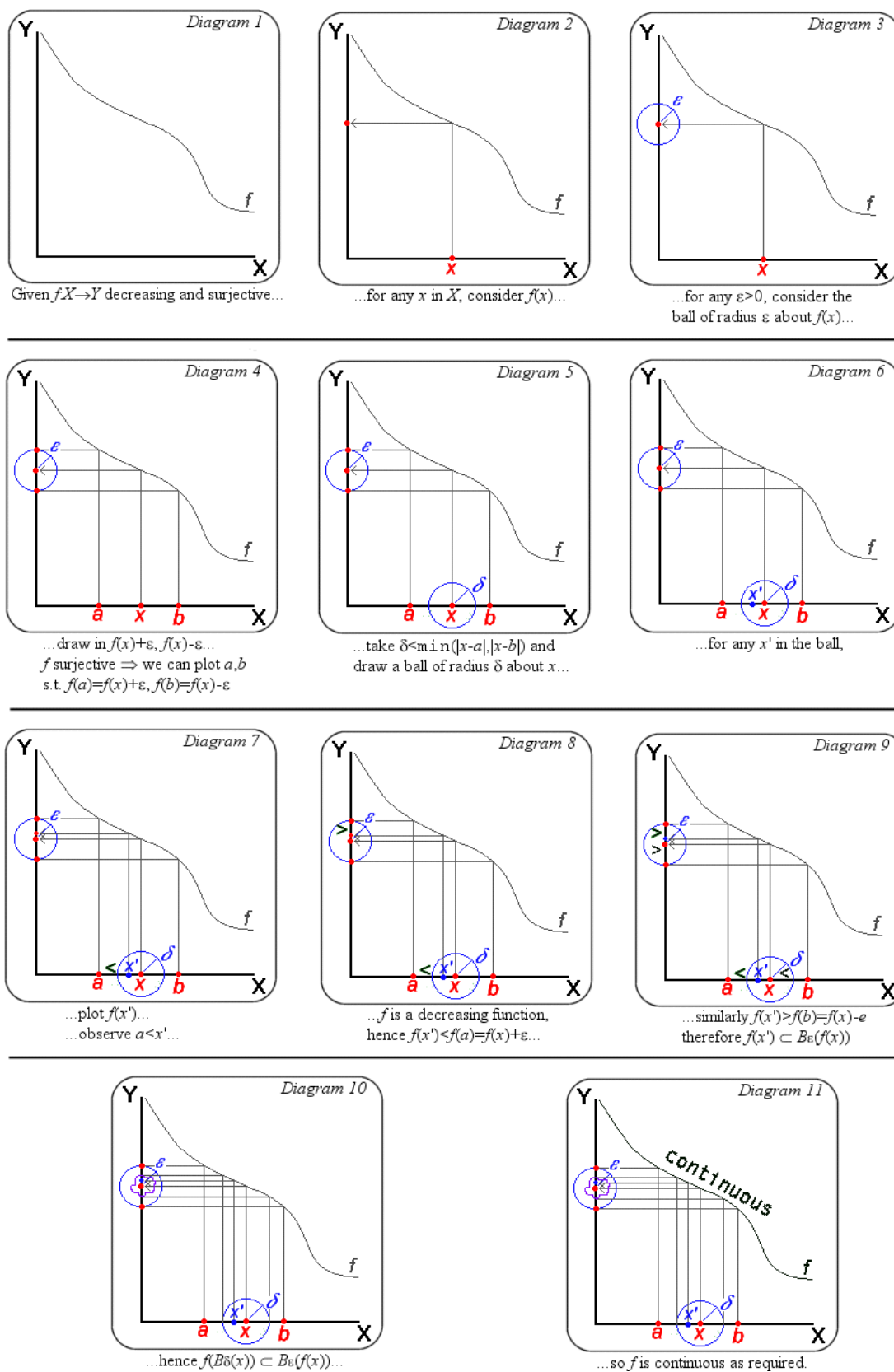


Figure 4.5. Diagrammatic proof that f surjective and decreasing $\Rightarrow f$ continuous.

$f(B_\delta(x)) \subset B_\epsilon(f(x))$. This is done by showing that for any point x' in $B_\delta(x)$ (diagram 6), we have $f(x')$ in $B_\epsilon(f(x))$ (diagram 9), hence $f(B_\delta(x)) \subset B_\epsilon(f(x))$ (diagram 10). We have now shown that for any $x, \epsilon, \exists \delta$ such that $f(B_\delta(x)) \subset B_\epsilon(f(x))$, which is the behaviour of a continuous function, hence we are finished (diagram 11).

Presentational issues

The comic-book style chains of diagrams used above are a cumbersome way of presenting a proof. This is because DDL is designed to be presented on a computer. On a computer (or a blackboard) the diagrams can be modified in situ, which avoids large chains and makes the steps much clearer. Another presentational problem is that the diagrams in Figure 4.5 become quite cluttered, which can hinder understanding. There are several things that can be done to improve this, all suited to a computer implementation. Object labels could switch from being drawn as normal when there is sufficient space, to popping-up only when pointed to (e.g. with the mouse) in crowded sections. This would allow all objects to be labelled. Observations such as `continuous(f)` can then be separated from the graphical objects and expressed algebraically (e.g. in a space at the bottom of the diagram) using the labels. Also some objects are not needed all the way through the proof (e.g. the points x' and $f(x')$ in diagram 10 of Figure 4.5), and can be made to disappear once they have been used.

4.1.2 Algebraic proof of the theorem

We now give an algebraic proof of this theorem for comparison. This proof has the same structure, but a very different feel.

Continuity Definition: Given $f: X \rightarrow Y$, $\forall x \in X$, $\forall \epsilon > 0$, $\exists \delta > 0$ such that $f(B_\delta(x)) \subset B_\epsilon(f(x)) \Rightarrow f$ continuous

Lemma: Let $X = (0, \infty)$, $f(x) = 1/x$, then $f: X \rightarrow X$ is decreasing and surjective

Proof:

1) f is surjective

$\forall a \in X, \exists b = f(a)$

$a.b = 1 \Rightarrow b.a = 1$

therefore $f(b) = a$

2) f is decreasing

$\forall x_1, x_2 \in X$ such that $x_1 < x_2$, let $y_1 = f(x_1)$, $y_2 = f(x_2)$.

Suppose $y_1 < y_2$

Let $d = x_2 - x_1$ then $y_2 \cdot (x_1 + d) = y_2 \cdot x_2 = 1$

$y_1(x_1 + d) < 1$

$y_1 \cdot x_1 = 1$ therefore $y_1 \cdot d < 0$

But $y_1 > 0$, $d > 0 \Rightarrow$ contradiction, so $y_1 \geq y_2$ as required.

Lemma: $f: X \rightarrow Y$ monotonic, surjective $\Rightarrow f$ continuous

Proof:

Without loss of generality, say f is an increasing function.

$\forall x \in X$, $\forall \epsilon > 0$, let $a = f(x) - \epsilon$, $b = f(x) + \epsilon$

f surjective $\Rightarrow \exists a', b'$ such that $f(a') = a$, $f(b') = b$

let $\delta = \min(|a' - x|, |b' - x|)$

f increasing, $a < y < b \Rightarrow a' < x < b'$

$\forall x' \in X$, $|x - x'| < \delta \Rightarrow a' < x' < b'$

f increasing $\Rightarrow f(a') < f(x') < f(b')$

$\Rightarrow |f(x') - f(x)| < \epsilon$

So $\forall x \in X$, $\forall \epsilon > 0$, $\exists \delta(x, \epsilon)$ such that $|x - x'| < \delta \Rightarrow |f(x) - f(x')| < \epsilon$, therefore f is continuous on X as required.

4.1.3 Summary of example proof

This example shows how diagrams can be used to give a rigorous non-trivial proof (although we have not yet explained the representations used or the mechanics of the reasoning). Comparing this proof with the corresponding algebraic proof shows both advantages and disadvantages for each method (bearing in mind that the theorem was chosen to suit diagrammatic reasoning, and so this is a biased comparison). The algebraic proof is more compact, but considerably drier. Both involve complex and technical ideas, and require some training to be understood.

4.2 Dynamic diagram logic

Here we discuss issues raised by the example proof in §4.1, and this leads to an explanation of how DDL works.

4.2.1 Representation of relations

Facts or *relations* in our diagrams are represented in several ways. Some, such as $x \in X$ are implicit in the way the objects have been drawn. These we call *implicit relations*. To capture this, our logic gives rules for spotting these relations, such as “If a point x is drawn inside a set X , then the diagram represents the relation $x \in X$ ”. Other facts are represented by a graphical annotation (e.g. $x = f(x)$ is represented by an arrow) and some are stated algebraically (e.g. $\text{continuous}(f)$ in Figure 4.4). We call these last two categories *explicit relations* – they must be explicitly stated since drawing the objects alone would not represent them. These concepts are formally defined in §4.3.2.1.

4.2.2 The structure of the proof

Note that the diagrammatic and algebraic proofs have very similar structures. This is not always the case though. For example, we assumed here that f is a well defined function; the diagrammatic and algebraic proofs of this *are* quite different – as shown in Figure 4.6.

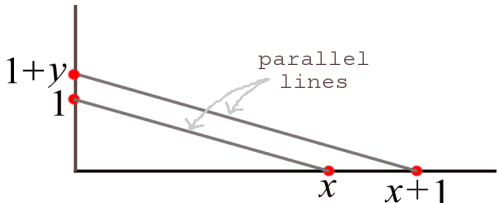
<u>Algebraic Proof</u>	<u>Diagrammatic Proof</u>
Depends on the approach taken to defining \mathbb{R} :	Below is a ruler & compass construction of $1/x$ from x given a unit length:
<i>Axiomatic</i> : True by definition, since \mathbb{R} is a field.	
<i>Constructive</i> : Depends on how \mathbb{R} is constructed.	$1:y = x:1$ Therefore $y = 1/x$

Figure 4.6. Differing approaches to proving that $f(x)=1/x$ is well defined.

4.2.3 Generalisation in diagrammatic proof

The example proof gives several examples of the generalisation process. Consider the lemma that $f(x)=1/x$ is a decreasing function (Figure 4.3). The diagrammatic proof used two specific points - say, $x_1 = 1.2$, $x_2 = 1.8$ - and showed f was decreasing in their case. However it did not depend on the specific values of the points drawn – indeed, their precise values

were not known. Instead the reasoning drew on the ('general') facts that $x_1, x_2 \in (0, \infty)$, $x_1 < x_2$ and that $\forall x, x.f(x)=1$. Thus the same reasoning could be applied to any two points in $(0, \infty)$, so we can conclude that the function $f(x) = 1/x$ is decreasing. However the reasoning could not be applied to any other function – because the reasoning requires $\forall x, x.f(x)=1$ – so we cannot generalise to other functions.

In order to avoid a dependence on the specific values of x_1 and x_2 , we cannot simply observe that $f(x_1) > f(x_2)$, but have to consider a case split. Diagrammatic case splits are slightly unusual: in the algebraic version, $f(x_1), f(x_2)$ are abstract points and their relative sizes are unspecified, whereas in the diagrams $f(x_1), f(x_2)$ have values (from where they're drawn) which are already fixed by this stage in the proof. So the diagram is already in one of the cases - $f(x_1) > f(x_2)$ - and the others are false for the diagram drawn. Hence performing a case split requires generating new models (for this case, we used new values for $f(x_2)$). We could create a model for each possible case (with respect to the relationships we consider, such as $x > y$ or $a \subset b$) when drawing each object. However this would quickly produce an unmanageable number of cases. Instead, we introduce the case split at the same point as in algebraic reasoning - that is, when a relationship is considered. This produces far fewer case splits. The '<' symbol is used as a label to mark relationships that have been 'observed', are known to be true for all cases in that branch of the proof program, and thus may be used in the reasoning.

The final section of the proof - that decreasing and surjective imply continuous - involves several generalisations:

- 1) generalisation from $x' \in B_\delta(x) \Rightarrow f(x') \in B_\epsilon(f(x))$ to $f(B_\delta(x)) \subset B_\epsilon(f(x))$
- 2) generalisation from $f(B_\delta(x)) \subset B_\epsilon(f(x))$ to the observation that this is true for all (positive) values of ϵ (and therefore f is continuous at x)
- 3) generalisation from f continuous at $x \in X$ to f continuous on X

Here generalisation must be carried out at steps during the proof as well as at the end of the proof. This is always the case for rules that recognise properties. Our approach to formalising such rules is to use 'animated' antecedents. These have a chain of diagrams as their pre-condition to capture the fact that the pre-condition is a type of behaviour.

Summary of generalisation method

- 1) Reason with specific instances.
- 2) Analyse the reasoning to check what cases it can be applied to.
- 3) Generalise to all cases where the chain of reasoning is guaranteed to be valid.

Such proofs, where the proof is a method that can be applied to all cases, are called *schematic proofs* [24]. An advantage of this approach is that it is not necessary to specify the correct generalisation in advance. Instead we can extract it from analysing the proof. This allows us to explore a theorem, discovering its limitations through our proof of it. This can be seen as a simple form of Lakatos's method of 'strategic withdrawal' [28]. It is possible because of the link between diagrams and models. It may be that a conjecture is false generally, but true for the case being considered. In this situation, we can apply the method of strategic withdrawal, and 'fall back' to a more restricted statement of the conjecture which we can prove.

4.2.4 Introducing redraw rules

Our examples in this section will be based on the property of being an open set. This is defined as follows:

Definition 4.2.4.1: If X is open...

$$\text{open}(X), x \in X \Rightarrow \exists \epsilon > 0 \text{ such that } B_\epsilon(x) \subset X$$

Definition 4.2.4.2: X is open if...

$$\forall x \in X, \exists \epsilon > 0 \text{ such that } B_\epsilon(x) \subset X \Rightarrow \text{open}(X)$$

DDL is defined using redraw rules, which are similar to rewrite rules but transform diagrams rather than formulae. This reflects our belief that diagrammatic reasoning is often linked to the drawing process, rather than just the finished diagram. These rules are expressed diagrammatically by an example transformation. A *simple redraw rule*, $D_0 \mapsto D_1$, consists of an initial diagram (D_0 , the antecedent or pre-condition) and a modified diagram (D_1 , the consequent, or post-condition). Figure 4.7 gives an example redraw rule. The antecedent is the top diagram, the consequent the bottom, and we use the convention that a shape drawn with a dotted line is interpreted as an open set. The antecedent diagram will match any point y in any open set Y ; the consequent guarantees the existence of a ball $B_\epsilon(y) \subset Y$.

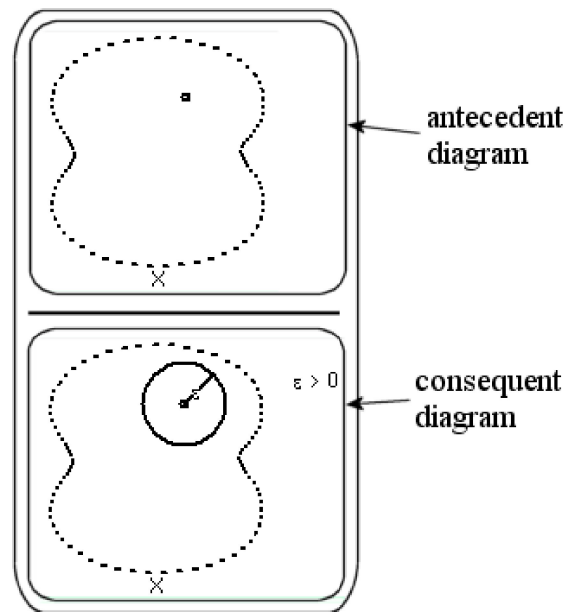


Figure 4.7. Definition 4.2.4.1 as a redraw rule.

Theorems are stated as rules and are expressed using the same diagrammatic representation. A proof consists of a demonstration that the theorem antecedent can always be redrawn to give the consequent diagram using an accepted set of rules (i.e. the axioms). Hence a proof is a chain of diagrams, starting with the theorem antecedent and ending with the theorem consequent. We refer to an incomplete or complete proof as a *proof program*.

Informally, the procedure for applying a simple rule is:

- 1) The antecedent diagram is matched with some part of the current diagram.
- 2) The current diagram is modified in an equivalent way to the modification between the antecedent and consequent diagrams. This modified diagram is added to the end of the reasoning chain, and becomes the new current diagram.

The principal differences from natural-deduction rewrite rules are:

- There can be an infinite number of valid (but equivalent) redrawings for a given diagram, a given rule and a given matching (e.g. a rule may specify that a point should be drawn, but leave open the choice of which point to draw).
- Due to the problem of multiple possible generalisations, there is no clear choice for how the matching algorithm should work.

4.2.5 Sometimes a diagram says too much

Our diagrams are drawings of specific cases, annotated with relational information. However some relations are represented 'implicitly', simply by being true in the model (e.g. $x \in X$ in Figure 4.7). Let us call this a 'model-based' approach to diagrammatic representations.

Consider trying to represent $x \in X \cup Y$ within this framework. Whatever model we choose to represent $x \in X \cup Y$ will also represent either $x \in X$ or $x \in Y$. To get round this, we introduce algebraic statements that cancel such unwanted relations. This is done by stating that the unwanted relation $r(x,y)$ – although true in the drawing – isn't *known* to be true, i.e. $\text{unknown}(r(x,y))$. This will be illustrated in §4.2.6.

Note that some `unknown` statements *will* be generally true (i.e. true for all cases) as a result of drawing constraints imposed by the other objects used in their construction.

As a result `unknown` statements (which should be automatically generated by any implementation) can sometimes be useful as a heuristic. Properties that need to be proved as sub-goals in a proof will sometimes appear early on in the proof as `unknown` implicit relations. Targeting these advances the proof. Hence the `unknown` statements can be used as a heuristic, focusing the user's attention on potentially key parts of the diagram.

4.2.6 Using branching for disjunction

DDL uses branching to create multiple diagrams for representing and reasoning with disjunction, as discussed in §4.2.3. Case splits are handled by splitting the proof program, with one branch for each case. Note that the decision on which case to draw is made when an object is created – that is, *before* the case split, and this drawing does not cover all the cases that might later arise. Hence when performing a case split, we must either generate modified diagrams covering the other cases, or work with inaccurate diagrams.

Figure 4.8 shows these two options for the case split $x \in X \cup Y \Rightarrow x \in X$ or $x \in Y$. In the left hand rule, we start with a model for the case $x \in Y$ (note the use of the `unknown` statement described in §4.2.5 to cancel this observation). After the case split, we keep this model for the $x \in Y$ case (whilst dropping the `unknown` statement), and we generate a modified model to

fit the case $x \in X$. In the right hand rule, we keep the same model for both cases. This means using algebraic annotations to (a) recognise the $x \in X$ case, and (b) cancel out the unwanted and $x \in Y$ relation which is still present.

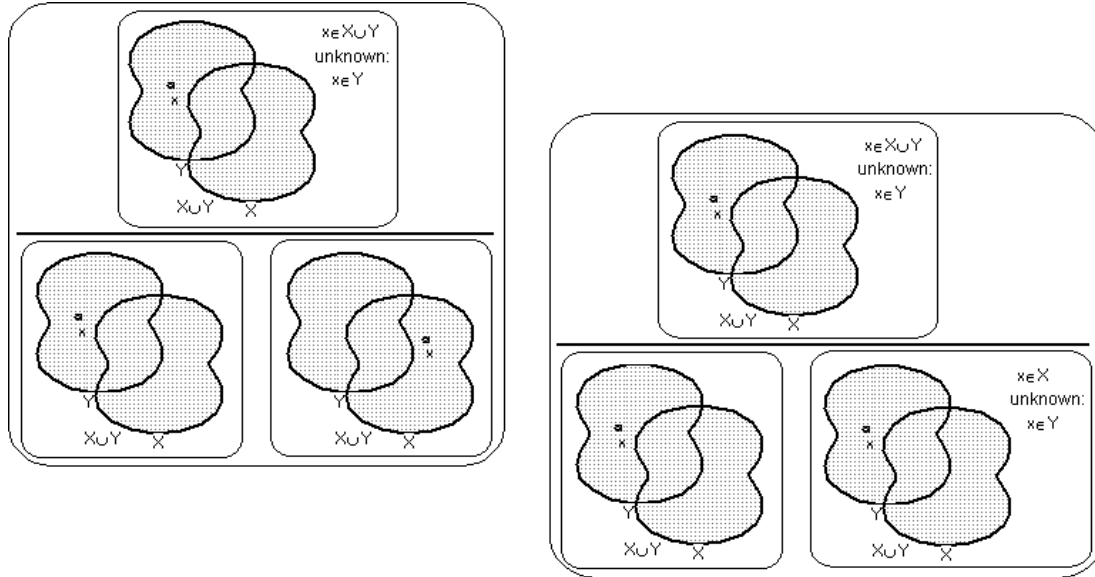


Figure 4.8. The $x \in X \cup Y \Rightarrow x \in X$ or $x \in Y$ branch rule with (left) and without (right) modifying the underlying model. Note that the intersection is not present as an object.

Clearly, generating a modified model (the left hand rule in Figure 4.8) is preferable in terms of producing understandable diagrams, since inaccurate diagrams are likely to be misleading. However there are *inconsistent* cases where an accurate diagram is impossible – the case split in Figure 4.3 is an example of this. In such cases we are forced to choose which relations will be inaccurate. We could automate this choice by ordering the different types of relation, with a preference to making the more visually striking relations accurate (e.g. in Figure 4.3, we chose a diagram that shows $f(x_1) < f(x_2)$ accurately at the cost of making $f(x_2) = “f \text{ applied to } x_2”$ inaccurate).

4.2.7 Quantifiers

Using animation for quantification

Figure 4.7 shows a diagram for “If X is open...”. Consider implementing the converse rule (definition 4.2.4.2). This definition can be read as “If, given any point x in X , we can find an $\epsilon > 0$ such that $B_\epsilon(x) \subset X$, then X is open”. Note the verb phrase ‘we can find...’ – this condition

can be thought of as dynamic: it gives a type of behaviour which we must demonstrate to show that X is open (by contrast, the condition “ X is open” in definition 4.2.4.1 can be thought of as *adjectival*). Static diagrams are not well suited to representing behaviour. They are better suited to adjectives than verbs. Instead, we introduce *animated redraw rules*. An *animation* here is a chain of diagrams. Animated redraw rules have an animation as their pre-condition. Where as simple redraw rules need only match the last diagram in the reasoning chain, animated rules must match a section of the reasoning chain. Figure 4.9 shows how we represent definition 4.2.4.2 as an animated redraw rule. The terms *strict* and *flexible* have not yet been defined. They describe the transitions in the animation, and will be used to distinguish between universal quantification (the point in Figure 4.9) and existential quantification (the ball in Figure 4.9). How this works is explained later in this section.

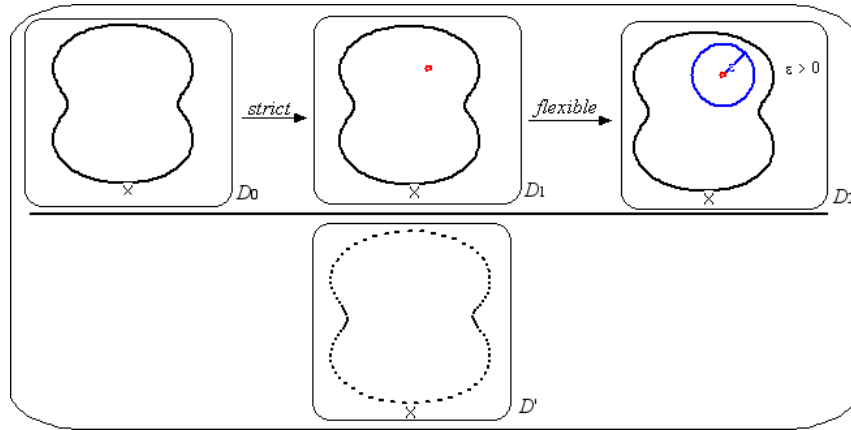


Figure 4.9. Definition 4.2.4.2 as an animated redraw rule.

Quantifier hierarchy

As with sentential reasoning, quantifier order can be important. Animation gives a reliable and intuitive ordering without introducing extra notation. This is because of causality: it is obvious that object A cannot depend on object B if B was drawn after A .

Generalisation

In DDL, a matching algorithm allows redraw rules specified in terms of specific examples to be applied to a wide range of diagrams. Thus the matching algorithm determines the generalisation of the rule (and vice-versa: specifying generalisations would establish a matching criterion). As discussed in §2.1.3, there are often several possible generalisations,

hence several matching algorithms are possible. These matching algorithms differ over which aspects of a diagram are important (and should form part of the matching conditions), and which should be ignored (i.e. generalised over). It seems unlikely that there will be a canonical answer to the question of what aspects of a diagram should be considered important. The difference is that conditions in algebra are explicitly stated, whilst diagrams often contain a lot of information that may or may not have been intentional.

We could simply make all the relevant information explicit in the diagram, and assume everything else is unimportant. However this would make for cluttered, less legible, diagrams. A more sensible approach is to have a default interpretation that certain aspects of a diagram are assumed to be important. Ideally, this should be the same as the intuitive reading of the diagram. The conditions specified by this default interpretation can be changed, but only through explicit conditions in the diagram. Spider diagrams show how this can be used for representing set membership, where spiders are used to override the default reading. The idea of default interpretations with explicit corrections can be extended to cover other relations. Some of the default readings we use are:

- 1) Lengths are considered unimportant (to be generalised), unless a statement of the form $\text{length}(a) < \text{length}(b)$ or $\text{length}(a) = n$ is present.³⁵
- 2) Set membership is considered salient, unless a statement of the form $\text{unknown}(a \in A)$ is present.

Default interpretations/explicit corrections are implemented in DDL using *implicit* and *explicit* relations, which are defined in §4.3.2.1. The full list of default interpretations for DDLA is given in §4.4.5.

Quantifier behaviour

We will now look at how quantifiers should behave in a redraw rule logic. Statements in DDL are expressed as rules, hence the question of “how do quantifiers in a rule antecedent behave?” becomes “when should a rule antecedent match a reasoning chain?” That is, at the syntactic level, the question of “what does a diagram/animation *mean*?” is recast as “what diagrams/reasoning chains does it match?”.

³⁵ This particular design decision (to represent $a > b$ algebraically rather than implicitly) was probably a mistake. See §4.4.5 for a discussion of this.

We have to be careful working in direct diagrams, as a quantified object is also a specific example (because the diagram requires an example object to be drawn).³⁶ For example, when reasoning about an abstract universally quantified point, we must nevertheless draw a particular point, and this point will have properties that do not hold universally. However, as long as such properties are not used in the reasoning that follows, they will not affect the generality of the proof. The reasoning that follows would work for any point, so it does not matter which point was actually drawn. The specific case that is drawn comes to represent a class of equivalent cases. What matters is that the reasoning is generic. With indirect representations, this is automatically enforced by using generic objects; with direct representations the generality of the reasoning must be checked.

Consider again Figure 4.9, where there is a universally quantified point x in the middle of the rule antecedent. Suppose we wish to apply this rule to show that the unit ball $Y=B_0(1)$ is open. First we introduce an arbitrary point $y \in Y$ to match the point x in the rule antecedent. We still have further reasoning to do before the rule will match: we have to find an ε -ball about y that lies within the set Y . The reasoning that follows must be universally applicable, which means that it must not use the specific nature of the point y , only the fact $y \in Y$. For example, suppose we concluded $B_{0.1}(y) \subset Y$ with $y=(0.7,0.8)$ by examination of $B_{0.1}(y)$. The reasoning would be sound for this case (since $B_{0.1}((0.7,0.8)) \subset B_0(1)$ is true), but it would not apply to other values of y . Hence the rule – which requires that such an ε exists for any point – would not be applicable.

This leads us to the following method for reliably enforcing generic reasoning: suppose an animated redraw rule has the antecedent $D_0 - D_1 - \dots - D_n$, where the D_i are diagrams. When a universally quantified object is introduced into the proof, it must be done exactly as shown in the rule. The interpretation of the object introduced into the reasoning chain must be equivalent to the interpretation of the object introduced in the rule antecedent. If diagram D_i introduces a universally quantified object, we call the transition $D_{i-1} - D_i$ a *strict* transition, since it will only match a transition $P_j - P_{j+1}$ if $P_j - P_{j+1}$ shows equivalent modifications to $D_{i-1} - D_i$ and no other modifications (i.e. no extra constructions or conditions).

³⁶ To be precise, because of blurring effects, each object is interpreted as a small³⁷ class of specific examples.

³⁷ Uncountable, but small.

When the rule antecedent contains an existentially quantified object, all that must be shown is that some matching object can be constructed in the reasoning chain. How this is done does not matter.³⁸ Hence an existentially quantified object can be drawn in any manner using several redraw operations, since all we require for the rule antecedent to match is that some such object exists. If diagram D_j introduces a universally quantified object we call the transition $D_{j-1} - D_j$ a *flexible* transition. A flexible transition allows arbitrary other constructions to be drawn in the reasoning chain when moving from one diagram in the rule to the next.

For example, consider the theorem “ $B_r(x)$ is open”. Proving this takes 11 steps in DDLA, with the final step being to apply the rule from Figure 4.9. A sketch of this proof is given in Figure 4.10 (using the symbol \hookrightarrow for “redraws to”); the full proof and rules used can be viewed on the CD-ROM. We start with the set $B_r(x)$ (diagram P_0 in Figure 4.10), which matches the set X in diagram D_0 , Figure 4.9. The first step is to introduce an arbitrary point in $B_r(x)$ to match the universally quantified point in diagram D_1 , Figure 4.9. It then takes three steps to construct a suitable ε -ball (P_5 in Figure 4.10) and five more steps to show that it lies inside $B_r(x)$ (diagram P_{10}). All these steps are performed using simple redraw rules. Finally we can apply the animated rule shown in Figure 4.9 and conclude that $B_r(x)$ is indeed open. Let $P_0 - \dots - P_{11}$ be the proof. Then D_0 matches P_0 , D_1 matches P_1 and D_2 matches P_{10} , as shown in Figure 4.11.

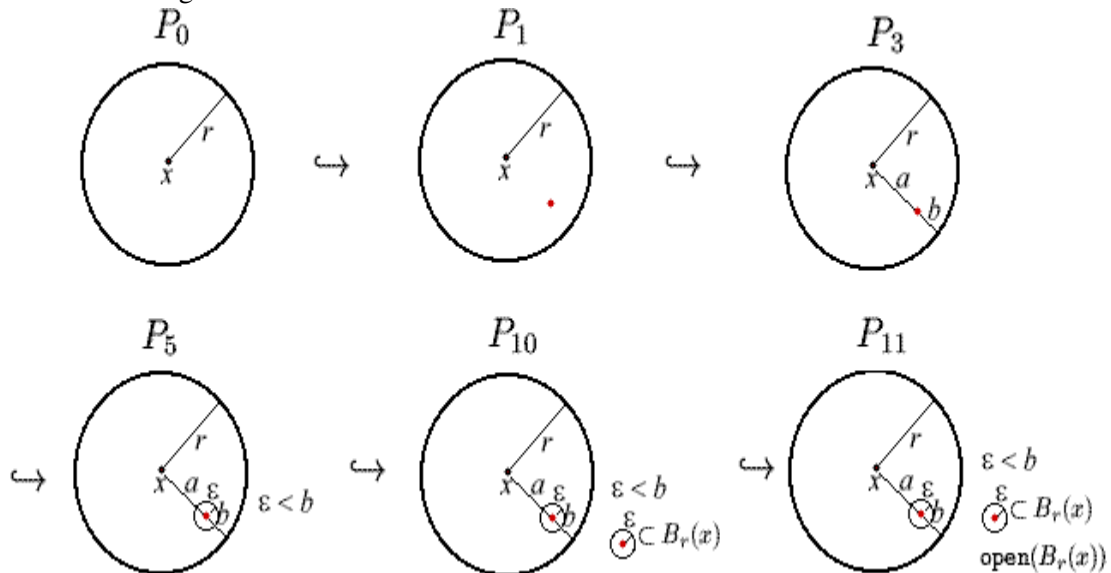


Figure 4.10. Sketch proof for “ $B_r(x)$ is open”.

³⁸ Provided new objects are created using existentially quantified rules that guarantee the existence of the objects they draw.

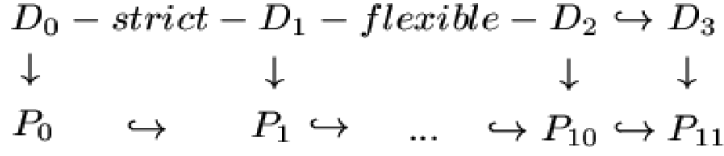


Figure 4.11. Antecedent matching with strict and flexible transitions.

Existential import

As discussed in §3.3.5, the natural way of reading a diagram such as D_1 in Figure 4.9 would probably be to assume that at least one point existed in the set. However definition 4.2.4.2 also applies to the empty set. There is thus a discrepancy between the natural reading of Figure 4.9 and the standard definition of an open set. The easiest way around this would be to keep the natural reading and treat objects such as the empty set as special cases, dealt with by their own rules. There are several drawbacks to such an approach though:

- 1) Our proofs would not carry over to cases where some of the sets involved are empty.
- 2) We would require more rules. Each concept defined would probably need an extra rule.
- 3) Our definitions would differ slightly from the standard ones.

It is mainly this last consideration that leads us to reject existential import. Using a logic with existential import would probably be wrong in a mathematics teaching tool, since it goes against convention and could cause confusion to students. Keeping existential import may be suitable to other domains though, especially those closer to real world problems.

Instead, we adjust the matching algorithm to allow vacuous quantification. We add a redraw rule for

$$X \Rightarrow \exists X \textit{ or } \textit{not}(\exists X)$$

We can then allow rules with $\forall X$ to match $\textit{not}(\exists X)$. This restores the standard mathematical meaning of universal quantification, although it is both less intuitive and complicates the mechanism of DDL.

Quantification in the rule consequent

In the rule consequent, all new objects are by default assumed to be existentially quantified. Universally quantified objects are explicitly identified, either by labelling or colour-coding. More complex consequents (i.e. those with mixed quantifiers) can be dealt with by breaking the statement into several rules. We could also use animated consequents for such statements. Such rules have not occurred yet in our work – although we do examine one in §8.2.1.

Quantifier scope

We limit DDL to representing a single nesting of quantifiers. That is, quantifier hierarchy in a diagram is a total order, and we cannot represent statements of the form “ $(\forall x.p) \Rightarrow (\exists y.q)$ ” where neither quantifier has scope over the other. This restriction (which has not caused any problems) could be removed by using the idea of dependent objects (c.f. definition 4.3.2) to define quantifier scope.

4.2.8 Aspects of DDL not illustrated in §4.1

Free rides and implicit inferences

'Free rides' are an important phenomenon in diagrammatic reasoning. A 'free ride' is when the representation for certain relations automatically implies some inferences without the need for explicit reasoning [45]. For example, if we represent $A \subset B$, $B \subset C$ in a diagram, we will automatically represent $A \subset C$. Free rides almost certainly add to the power of diagrammatic reasoning, and it would seem sensible for a diagram logic to take advantage of this. DDL does so, formalising these within the system via *implicit inference rules* (c.f. §4.4.6).

Emergent objects

Closely related to free rides is the phenomenon of *emergent objects*. These are diagram objects formed as a side-effect of drawing other objects. Examples would be the creation of set regions such as $A \cap B$, or recognising that the triangles in Figure 4.12 form two squares. If we are to reason about emergent objects, we need rules for recognising them. These rules need to both recognise emergent objects, and check that they always exist (i.e. that the emergent object is a necessary consequence of the properties of other objects in the diagram, and not merely an accident of drawing choices). This is not always simple. For example, the triangles in Figure 4.12 clearly form a square – c^2 – within a square – $(a+b)^2$ – but that the interior shape *must* be a square is less obvious (it draws on the fact that they are right angled triangles, and that their interior angles add up to 180°). Automatically verifying such facts could hide important aspects of the proof from the reasoner – which we would not want in an educational tool. So, whilst emergent objects undoubtedly add to the power of diagrammatic reasoning, it is not clear that we want to build them into our logic. A sensible compromise might be to automatically detect simple emergent objects, such as the formation

of sets like $A \cap B$ or that the outer square in Figure 4.12 is a square, but not emergent objects that require non-trivial reasoning to demonstrate their existence, such as the inner square in Figure 4.12.

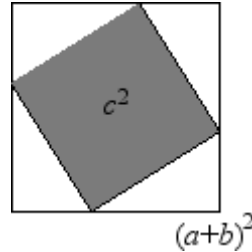


Figure 4.12. Emergent objects in Pythagoras' theorem

Reasoning with counterexamples

So far we have looked at proving general conjectures (i.e. theorems of the form $\forall x...$), and thus focused on the problems involved in correct generalisation. We also want to be able to disprove such conjectures when false by producing counterexamples. The strong link between models and diagrams suggests that diagrammatic reasoning could be well-suited to reasoning about counterexamples. We give one example below:

Suppose we wish to disprove the statement: $A \subset X \cup Y \Rightarrow A \subset X$ or $A \subset Y$ (which we represent in DDL as a redraw rule that can be added as an assumption to the axioms). Figure 4.13 provides a counterexample (since $B \subset X \cup Y$ but $B \not\subset X$ and $B \not\subset Y$).

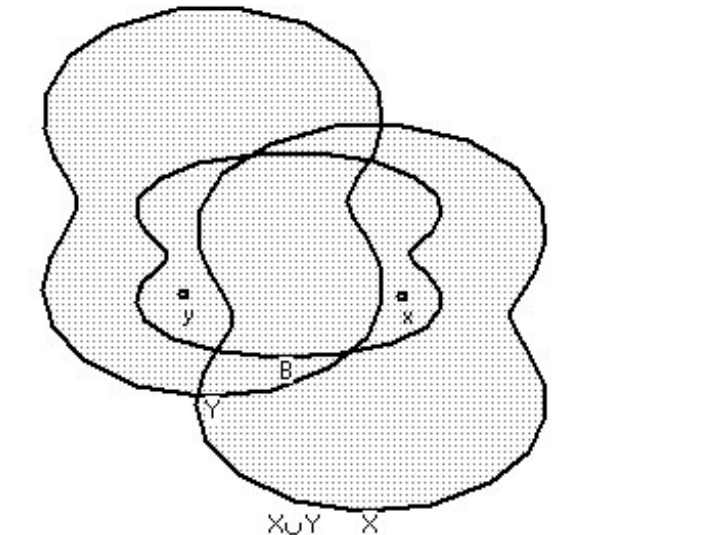


Figure 4.13. A counterexample to $A \subset X \cup Y \Rightarrow A \subset X$ or $A \subset Y$.

We would like to incorporate counterexample reasoning into DDL. It is helpful to think of such reasoning as taking place in two stages:

- 1) Finding/presenting the counter-example (& showing it exists).
- 2) Demonstrating that it is indeed a counter-example.

1) Presenting an example

Unlike the reasoning we have considered so far, this first stage doesn't involve using diagrams that generalise. Instead it requires the opposite approach – interpreting the diagram as referring to one specific example. That is, instead of reading graphic objects as “Given any line...”, we read them as “Consider *this* line *here*...”. Syntactically, this can be achieved by using an existential quantifier and fixing (making into observed relations) the graphic object's parameters. In doing so, we need to know that the diagram does indeed correspond to a model.

Moreover, the diagram will often contain several implicit relations that we would normally cancel using *unknown* statements because they are not generally true. In counter-example reasoning however, we want to assert these relations as being true for this case. For example in Figure 4.13, we draw $x \in B$, but also want $x \in X$ and $x \notin Y$, which is true for this case but not for all X, Y, x and B . In algebraic reasoning this would be equivalent to deducing “ $\exists x. x \in X, x \notin Y$ ” from “ \exists sets X, Y, B and point x such that $x \in B$ ” – which is not valid without first specifying values for x, X, Y, B and demonstrating the required relations. The ability to perform such steps is one advantage of diagrammatic reasoning. For it to work, we need to be able to reliably link graphic objects with domain objects that share the same relations. This will work – thanks to the close link between diagrams and models – as long as the diagram is *accurate* (which we will define in §5.1.2) and our algorithms for evaluating implicit relations are reliable. Unfortunately, we cannot generally evaluate whether or not a diagram is accurate. As discussed in §3.1.3, a diagram can contain inaccuracies too small to spot. This will limit our use of such shortcuts to \subset and \in relations, where we can reliably evaluate the accuracy of the diagram.

2) Demonstrating that an example is indeed a counterexample

Having found a suitable example, we may still need to do considerable work to show that it is indeed a counterexample. This second stage consists in adding the conjecture as an assumption, and showing that this leads to a contradiction when applied to the

counterexample. Note that this work may involve using arbitrary (universally quantified) graphic objects again. For example, to show a set X is not open, we must show $\exists x \in X$ such that $\forall \epsilon > 0. B_\epsilon(x) \subset X \Rightarrow \text{false}$.

Our Method

We accomplish the first stage by having two sets of basic object-creation redraw rules: one that draws universally quantified objects (to be used for proofs such as in §4.1), and one that draws specific objects (existentially quantified and with their parameters as observed relations) to be used in creating counterexamples.

This gives the following method for disproving a conjecture (provided it can be expressed as a redraw rule) by counter-example:

- 1) Add the conjecture as a redraw rule R to the usual axioms.
- 2) Draw a diagram using specific-object redraw rules
- 3) Use R and other rules to derive a contradiction.

(where steps (2) and (3) can be intermingled)

Such a proof is finished when all the branches of the reasoning program have been shown to contain a contradiction.

Cleaning up the diagram

As observed in §4.1.1, diagrams can become too crowded, hindering understanding. Some objects are needed during the reasoning, but do not form part of the theorem. Once such an object has served its purpose, it can be deleted without affecting the reasoning. This is equivalent to removing variables created in sub-goals in a system such as Prolog. However in DDL we do not have separate logical spaces for sub-goals; all reasoning is performed at the same level. To do these 'cleaning up' jobs, we have a redraw rule 'delete-object' which is applied automatically as follows: If a redraw rule deletes an object, we automatically apply delete-object to all *dependent* objects. That is, if a redraw rule R deletes an object X (as happens in Figure 4.25, where both the point and the ball are deleted), we also delete those objects created using X , and which usually have little meaning without X .³⁹ For example, suppose we have created a point x , followed by the point $f(x)$. If we delete x , we would then also delete the point $f(x)$. Object dependency is formally defined in §4.3.2.

³⁹ This is usually the desired behaviour. However there are cases where it is desirable to delete more objects: sometimes intermediate objects, which are not of interest and could be deleted, are created during the construction of some final object which we do not want to delete.

Note that deleting objects involves *throwing away* information. It is therefore always sound to delete an object, since this can only hinder the application of other rules (i.e. deleting an object cannot allow a rule that shouldn't apply to apply). This is because we do not attach any meaning to white-space, and hence cannot have rules that require the absence of an object.⁴⁰

Deleting objects will inevitably leave behind relations which are no longer meaningful. To deal with these we have a rule `delete-relation` which is automatically applied to relations that refer to non-existent objects (i.e. given diagram D , we delete $r(x_1 \dots x_n)$ to give diagram D' if $\exists i$ such that $x_i \notin \text{labels}(D)$). Technically, this is not a redraw rule (because we do not have a matching for x_i). However it is sound, since again it only involves throwing away information.

4.3 Formalising DDL

Our strategy in formalising DDL will be to separate diagrammatic reasoning into two processes:

- 1) Parsing diagrams, when the different objects are identified and relations about them are extracted from the diagrammatic representation.
- 2) Rule application and generalisation. Having separated this from the parsing stage, we can specify it at an algebraic level (where diagrams are considered to be sets of abstract objects and relations).

There are two elements of DDL that do not fit this framework however: the use of diagrams-as-models in counter-example reasoning, and the use of flexible-matching/condition-updating in exploring a conjecture. Formalising these aspects of DDL requires extra machinery which will relate reasoning steps directly with the diagrammatic representation. Eventually, this will involve us in a detailed examination of the representations that reaches down to the level of how our diagrams are drawn (c.f. §5.1.1).

4.3.1 Notation

We use the following notational conventions. Many of the terms listed here have not yet been defined, but will be defined in this chapter. We have grouped all our notational

⁴⁰ Other diagram logics may wish to actively use white-space. In such a logic, deletion would not necessarily be valid.

conventions in one place for convenient reference. There is also a glossary of terms in §10 which may be helpful.

- Upper case letters (sometimes with subscripts or primes) are used for diagrams, redraw rules and (within the context of a specific diagram) sets.
- Underlined upper case letters are used for reasoning programs.
- In the context of reasoning programs, let $D - D'$ denote that D' is a child node of D .
- Lower case letters (sometimes with subscripts or primes) are used for graphic objects.
- Relations are written in the form $r(x_1, x_2, \dots, x_n)$ where r is the relation name and x_i are object names or constants. Relations may have any arity. However, since binary relations are the most common, we will allow $r(x, y)$ to represent an arbitrary relation.
- \Rightarrow has its standard logical meaning of “implies”.
- \simeq means “matches” e.g. “ $A \simeq B$ ” is used for A matches B
- $\simeq_m \simeq$ means “matches with mapping m ”. Mapping functions are overloaded so that the same name designates several different functions depending on the type of input.
- \mapsto means “redraws to” e.g. “ $D \mapsto D'$ ” is used for “ D redraws to D' ”
- $\overset{R}{\mapsto}$ means “redraws using rule R ”

4.3.2 Diagrams

We define a diagram in DDL to be a set of *graphic objects* and a bag⁴¹ of *observed relations* – portrayed graphically or algebraically. We write this as $D=(objects, relations)$.

Meta Functions

We define the functions:

```
objects: $D \rightarrow \{\text{graphic objects in diagram } D\}$ 
relations: $D \rightarrow \{\text{observed relations in diagram } D\}$ 
label: $object \rightarrow object\ label$ 
labels: $D \rightarrow \{\text{labels used in diagram } D\}$ 
```

These functions uniquely define a diagram (in that we have defined a diagram to be a set of labelled objects and relations, and these functions give access to all these different parts of the diagram).

⁴¹ Unlike a set, a *bag* (also called *multi-set*) may contain repeated elements.

We will use several notational shortcuts. If A, B are diagrams, we write “ $\text{objects}(A \setminus B)$ ” for “ $\text{objects}(A) \setminus \text{objects}(B)$ ”, “ $\text{relations}(A \setminus B)$ ” for “ $\text{relations}(A) \setminus \text{relations}(B)$ ” and “ $A \cap B$ ” for “ $(\text{objects}(A) \cap \text{objects}(B), \text{relations}(A) \cap \text{relations}(B))$ ”.

Objects

Graphic objects are physical symbols, drawn on any flat surface (e.g. a computer screen or a blackboard). All graphic objects must be labelled and have a clear type. Labels are created in any suitable language. They are first class objects (i.e. the structure of a label carries no meaning) and must be unique within the diagram. Objects that are identical except for labels are allowed (these would be drawn as if they were one object with multiple labels, but are treated as multiple objects). Object types are partially ordered (i.e. there are subtypes). Formally, types are treated as a kind of relation. We assume here that the user can (a) unambiguously separate out the different objects, (b) recognise the object's type, and (c) identify the correct label associated with each object. These assumptions will be discussed in chapter 5.

Dependent objects

The concept of dependent objects is used in object deletion (performed to improve the clarity of a diagram; c.f. §4.2.8) and also appears as a condition on counter-example reasoning (c.f. §4.3.5). Objects are dependent upon those objects used to create them, with dependency also being transitive. Object dependency is defined below:

Definition 4.3.2.1: Object dependencies

An object x is *dependent* upon object y iff:

- x was created by redraw rule $R: T \rightarrow T'$ with matchings m_1, m_2
and $y \in m_1(\text{objects}(T))$.
- or*
- x is dependent upon z and z is dependent upon y .

Write this as $\text{dependent}(x, y)$.

Constants

Constants are labels which can be used in relations without a corresponding object being present in the diagram. The statement “ $\epsilon > 0$ ” in Figure 4.7 gives an example of this – ϵ is an object (the radial line of the ball) whilst 0 is a constant. When matching diagrams, constants are mapped to themselves.

Relations

Relations are predicate statements ranging over graphic object labels and constants (e.g. $\text{open}(X)$ or $\text{fn}(f, x, y)$). Relations are divided into two basic categories: *observed relations* or *observations*, which are 'known facts' that the logic can use, and *unobserved relations*, which might be true in the diagram, but cannot be used in the reasoning. Typically, unobserved relations are either irrelevant to the theorem (and so should be ignored), or haven't yet been proved (and so cannot be used).

Observed relations are divided into *explicit* and *implicit relations*.

Unless explicitly overridden, implicit relations can be read from the diagram without having to be stated. They arise from the way the graphic objects are drawn, and are a key difference between diagrammatic and textual representations. An example is that we do not need to explicitly state relations of the form $X \subset Y$, since this information is implicit in drawing X inside Y . Implicit relations are read according to a set of rules of the form: “If $r(x, y)$ appears true in D and is not explicitly negated, then $r(x, y)$ is observed in D ”. We call these the *implicit relation rules*. These rules introduce a link between the semantics of the diagrams and the syntactic functioning of DDL, and require sound mechanisms for checking the truth of the relations involved. It is intended that such relations will represent visually intuitive concepts, but the mathematical semantics of a relation need not be simple. We introduce a meta-predicate `implicit` to state that a relation can be observed by the implicit relation rules. Note that the implicit relation rules operate at the level of *appearances*. This means that they must take into account the accuracy of the drawing/perception process. It also means we have to detect when appearances are unclear. In such cases, the relation or its negation (whichever is intended) is stated explicitly. We impose the condition on implicit relations that:

$\text{implicit}(r(x, y)) \Rightarrow r(x, y)$ is true for all domain objects x', y' such that x', y' are indistinguishable from x, y when drawn.

Explicit relations must be stated. They can be represented either graphically (such as using the right angle symbol \perp) or algebraically, using labels to reference objects. If an explicit relation contradicts an implicit relation, the explicit relation is taken to be true, and the implicit relation is ignored.

We also define the special relation *false* - denoting that the diagram represents a situation known/assumed to be false - the second order relation $\text{not}(r(x,y))$ – with the usual semantics – and a special meta-relation called *unknown*, used for cancelling unwanted implicit relations:

Definition 4.3.2.2: Unknown relations

$\text{unknown}(r(x,y))$ means $r(x,y)$ cannot be observed from the diagram.

Equivalently, $\text{implicit}(r(x,y))$ *without* $\text{unknown}(r(x,y))$ means

$r(x,y) \in \text{relations}(D)$

Implicit Inference Rules

As discussed in §4.2.5, when drawing an object, our choice of placement can unavoidably create implicit relations beyond those intended. Some of these extra relations are unwanted. However some of these unintended relations are beneficial. For example, if we draw A, B, C so as to represent the relations $A \subset B, B \subset C$, we will also represent $A \subset C$. We could annul this extra relation with the statement $\text{unknown}(A \subset C)$, but since it is a valid and obvious inference step, it would seem perverse to do so.

To distinguish between desirable extra relations and unwanted ones, we introduce *implicit inference rules*. These specify the free rides that can be generated. They must be stated in the specification of a DDL, and form part of the logic in the same way that the implicit relation rules do. For example, for the \subset relation, we have the implicit inference rules:

$$X \subset Y, Y \subset Z \Rightarrow X \subset Z$$

$$X \subset X$$

$$X \subset Y \Rightarrow f(X) \subset f(Y) \quad [\text{where } f \text{ is the set-to-set extension of a function}]$$

An extra relation is a free ride that should be kept if it can be derived from other relations using the implicit inference rules. Any other extra relations are possibly false and should be removed. Implicit inference rules are formally implemented as simple redraw rules. They are

different in that (a) they are triggered automatically and (b) they are not presented to the reasoner as separate steps in the proof.

Reasoning programs

A *reasoning program* is a rooted directed acyclic graph of diagrams with labelled arcs. Either reasoning programs are created by starting with a root node and successively applying redraw rules – in which case we say the program is *constructed* – or they occur as animated antecedents. In a constructed reasoning program, we label the arcs between nodes with the redraw rule and matching used to create that transition. These arc labels specify a sequence of diagram transformations. This sequence can be applied to other diagrams (specifically, any diagram which the program's root diagram matches) – hence the term 'program'.

A *sub-program* is a connected rooted sub-graph of a reasoning program.

A *proof program* is a reasoning program presented as proof of a conjecture.

A *reasoning chain* is a reasoning program where each node has at most one child node.

Let \underline{D} be a reasoning program. We define the functions:

$$\text{diagrams}(\underline{D}) = \{\text{nodes in } \underline{D}\}$$

$$\text{root}(\underline{D}) = \{\text{the root node of } \underline{D}\}$$

$$\text{leaves}(\underline{D}) = \{\text{leaf nodes (i.e. those without any outgoing arcs) in } \underline{D}\}$$

$$\text{rules}(\underline{D}) = \{\text{redraw rules } R \text{ used to construct } \underline{D}\} \text{ (only defined if } \underline{D} \text{ is constructed)}$$

In animated redraw rules, the antecedent is a reasoning chain, with each edge labelled as a *strict* or *flexible* transition (c.f. §4.2.7 for an discussion of this with an example). These two types of transition are used to control quantifier type: Objects introduced by strict links are universally quantified, those introduced by flexible links are existentially quantified. Any nesting of quantifiers is allowed. However the animated rules we have explored are all of the pattern: $R:A_1 \text{ -s- } \dots \text{ -s- } A_{n-1} \text{ -f- } A_n \Rightarrow A_{n+1}$ where *-s-*, *-f-* denote strict and flexible links respectively. The pre-condition for a rule of this form could be read as:

$$\forall X_1, \dots, X_m. (\text{conditions}(X_i) \Rightarrow \exists Y_1, \dots, Y_k. \text{conditions}(Y_j)) \Rightarrow A_{n+1}$$

In constructed reasoning programs, each link is taken to be strict. We sometimes wish to extract the quantification details from a reasoning program. This is done according to definition 4.3.2.3 below.

Definition 4.3.2.3: $Q(D|\underline{D})$

[Interpreting quantification in a reasoning program]

Given a diagram D in constructed reasoning program \underline{D} , define a function $Q(D|\underline{D})$ which returns the quantifiers of objects in D , nested according to the quantifier hierarchy in \underline{D} . Q is defined as follows:

Let $D_1 \dots D_n$ be diagrams such that D_1 is the root node of \underline{D} , $D_n = D$ and $D_1 \dots D_n$ form a chain in \underline{D} .

Then let $Q(D|\underline{D}) = Q(D_n|\underline{D})$

$Q(D_1) = \forall \text{labels}(D_1)$

$Q(D_i|\underline{D}) = Q(D_{i-1}|\underline{D})(Q(R).\text{labels}(D_i \setminus D_{i-1}))$, where R is the redraw rule used to create D_i , which must have a specified quantifier type (c.f. §4.3.3).

Diagram matching

Deciding which diagrams do and don't match is crucial to DDL. This is an area where diagrams are more complex than algebra. Diagram matching performs an equivalent task to unification and pre-condition checking in algebraic reasoning. For unification there is – at least in first order logic – a canonical algorithm once it has been decided what constitutes a variable. By contrast, it is not clear what elements of a diagram should be allowed to behave like variables and change when matching, nor what differences should be allowed. For example, consider the two triangles in Figure 4.14. Are they equivalent? Should they match each other? Would they match a pentagon? We defer such decisions to later, locating them in the decisions as to which relations can be represented implicitly. Definition 4.3.2.4 below is both simple and general. Depending on the set of implicit relation rules, it could be used in a variety of quite different logics.



Figure 4.14. Equivalent triangles?

Definition 4.3.2.4: Diagram matching

Say “diagram A matches diagram B ” if the following conditions hold:

\exists a function $m: \text{labels}(A) \rightarrow \text{labels}(B)$ such that $\forall r(x,y) \in \text{relations}(A)$, we have $r(m(x), m(y)) \in \text{relations}(B)$

[note: this includes type relations]

Write $A \simeq_m B$ for “ A matches B with mapping m ”

Let us illustrate this definition by considering Figure 4.14 again. First note that diagram matching is not symmetric. A diagram for 'any triangle' can match a right angled triangle, but not vice-versa. If right angles are implicit relations (i.e. can be inferred from the drawing) but other angles aren't, then $\text{relations}(A) = \{\text{triangle}(A)\}$ and $\text{relations}(B) = \{\text{triangle}(B), b_1 \perp b_2\}$, where b_1, b_2 are the non-hypotenuse sides of B . Hence A would match B , but B would not match A . However, if right angles are explicit relations, then $\text{relations}(B)$ is just $\{\text{triangle}(B)\}$, and so B would also match A .

Note: For reasons of convenience and clarity, we will 'overload' the matching functions so that they apply to objects as well as labels. If m is a matching from diagram A to diagram B , then we can define $m: \text{objects}(A) \rightarrow \text{objects}(B)$ by $m(a) = b \Leftrightarrow m(\text{label}(a)) = \text{label}(b)$.

The following definitions will be used in defining how redraw rules operate:

Definition 4.3.2.5: Diagrammatic Substitution

Say $D' = D[a/b]$ if $\text{objects}(D') = \text{objects}(D) \setminus \{b\} \cup \{a\}$ and $\text{relations}(D') = \{r' : r \in \text{relations}(D), r' = r[\text{label}(a)/\text{label}(b)]\}$ where $r[x/y]$ is r with all occurrences of x replaced by y

Note that implicit relations must be taken into account when checking that $\text{relations}(D') = \text{relations}(D)[a/b]$. Changing the object drawn can affect what implicit relations are generated. As a result, both the unknown statements and the explicit statements may have to be changed. Hence diagrammatic substitution is a more complicated procedure than substitution in algebraic reasoning.

4.3.3 Redraw rules

Redraw rules form the core of DDL. They are the visual equivalent of rewrite rules. Rules are defined by an example redrawing. This is coupled with a mechanism for applying the rule to matching diagrams/reasoning programs which we define below. A theorem is stated in the same form as a redraw rule (indeed, it is just a redraw rule that is not taken as an axiom). A proof is a sequence of redrawings demonstrating that the redrawing operation

shown in the theorem statement can be done using the accepted redraw rules. Although the proof only modifies one diagram, it constitutes a redrawing *program* that can be applied to any matching diagram.

We define four different types of first order (quantifying over diagram objects)⁴² redraw rules: simple rules, branch rules, contradiction rules and animated rules. Note that these four types can all be covered by one type of rule (animated branch rules). It seems more natural though to treat them separately, since there is a clear split in their usage. We also introduce three meta rules that are treated individually in §4.3.4.

Given a rule R that we wish to apply to a reasoning program \underline{D} , let D be the leaf diagram in \underline{D} that we wish to redraw, and D' the new diagram we create. We define the notation “target(R, \underline{D})” for D , and “ $R(D)$ ” for D' (note that R is not a function, so this notation only makes sense in contexts where D' has already been created).

Simple rules

A simple rule R consists of two diagrams – an antecedent T_1 and a consequent T_2 – where the consequent is a modification of the antecedent. We write this as $R:T_1 \mapsto T_2$. We define a function $Q:\text{rules} \rightarrow \{\forall, \exists\}$ by:⁴³

$$Q(R) = \{\forall \text{ if } R \text{ creates universally quantified objects, } \exists \text{ otherwise}\}$$

The result of applying a redraw rule will generally not be unique. Indeed, in many cases there will be an infinite number of acceptable redrawings. Therefore a rule cannot be applied without either the user or a heuristic choosing a valid redrawing.⁴⁴

Suppose we wish to apply a rule R to a diagram D_p in reasoning program \underline{D} , redrawing D_p to give diagram D_{p+1} (in extended reasoning program \underline{D}). If this redrawing is a valid application of R , then we write this as $D_p \xrightarrow{R} D_{p+1}$ (for a simple rule) or $\underline{D} \xrightarrow{R} \underline{D}'$ (for an animated one).

42 Since diagram objects include functions, this includes some statements that are 2nd order predicate logic statements.

43 As discussed in §4.2.7, our rules create either universal or existential objects, but not both.

44 Note that an acceptable redrawing will not necessarily be a *good* (i.e. clear and easy to understand) redrawing.

Definition 4.3.3.1: Simple rule application

Let changes be the function:

$\text{changes}(A,B)=(\text{labels}(B \setminus A), \text{labels}(A \setminus B), \text{relations}(B \setminus A), \text{relations}(A \setminus B))$
i.e. $\text{changes}(A,B)$ lists the objects created and deleted, and the relations created and deleted, when moving from diagram A to diagram B .

$D_1 \mapsto D_2$ is a *valid application* of rule $R:T_1 \mapsto T_2$ if the following conditions hold:

(a) Matching: $\exists m_1, m_2$ such that $T_1 \simeq m_1 \simeq D_1$ and $T_2 \simeq m_2 \simeq D_2$ and $m_1=m_2$ where defined

Equality of Changes:

(b) $\text{changes}(D_1,D_2)=\text{changes}(m_1(T_1),m_2(T_2))$, where $m_i(T_i)$ is the diagram T_i with all labels changed to their images under m_i

Branch rules

Branch rules handle case splits by splitting the reasoning program into multiple branches, one for each case.

Definition 4.3.3.2: Branch rule application

$D_0 \mapsto D_1 \dots D_n$ is a valid application of rule $R:T_0 \mapsto T_1 \dots T_n$ if

$\forall i \in \{1 \dots n\}, D_0 \mapsto D_i$ is a valid application of $R_i:T_0 \mapsto T_i$ as in definition 4.3.3.1.

Contradiction rules

The opposite of introducing a case split is to eliminate a case by showing it cannot exist – i.e. by finding a contradiction. We introduce the symbol “false” to signify a contradiction. false is defined as a graphic object, and contradiction rules are handled as a type of simple rule. The absence of variables that quantify over relations means we need multiple contradiction axioms (i.e. for each relation R , we need a separate rule $\{R \text{ and } \text{not}(R)\} \Rightarrow \text{false}$).

Animated rules

An animated rule has a reasoning chain as its precondition, with each link labelled as either strict or flexible (c.f. §4.2.7 for an informal description). For simplicity, we restrict animated rule consequents to adding relations and performing deletions (since this is all that we have needed in our work, and possibly covers all cases where we would want to use animated rules). Before defining how these rules operate, we must first define matching

between two reasoning programs. There are two parts to this definition: one for 'normal' matching, and one to handle vacuous quantification, as discussed in §4.2.7.

Definition 4.3.3.3: Program matching I

Say “reasoning program \underline{A} matches reasoning program \underline{B} with mapping m ” if \exists functions $m:\text{labels}(\underline{A})\rightarrow\text{labels}(\underline{B})$ and $m:\text{diagrams}(\underline{A})\rightarrow\text{diagrams}(\underline{B})$ (overloading the matching function again) such that:

- (a) \forall diagrams $A\in\underline{A}$, \exists diagram $B\in\underline{B}$ such that $A\simeq_{m_A}B$, where $m_A=m$ restricted to A .
- (b) \forall diagrams $A,A'\in\underline{A}$ let B,B' be the diagrams they match. If A' is a child of A then B' is an ancestor of B .
- (c) \forall diagrams $A,A'\in\underline{A}$ such that $A-A'$ is a strict link in \underline{A} , then let B,B' be the diagrams they match and N be the simple redraw rule $N:A\rightarrow A'$.
We require $B\stackrel{N}{\mapsto}B'$ as defined in definition 4.3.3.1.
- (d) \forall diagrams $A,A'\in\underline{A}$ such that $A-A'$ is a flexible link in \underline{A} , let B,B' be the diagrams they match and $N_1\dots N_k$ be the redraw rules used to draw B' from B .
Then we require $Q(N_i) = “\exists”$ for all $i\in\{1\dots k\}$.
- (e) If A is the leaf node of \underline{A} , then $m(A)\in\text{leaves}(\underline{B})$

Note: condition (c) in definition 4.3.3.3 is what ensures the chain of reasoning is as general in reasoning program \underline{B} as it is in program \underline{A} . It also means that a strict link in a reasoning program defines a redraw rule (essentially, a strict link *is* a simple redraw rule).

The matching process defined above is loosely illustrated in Figure 4.15. This shows the animated pre-condition for recognising an open set (c.f. Figure 4.9) being matched with the diagrams of a target reasoning program.

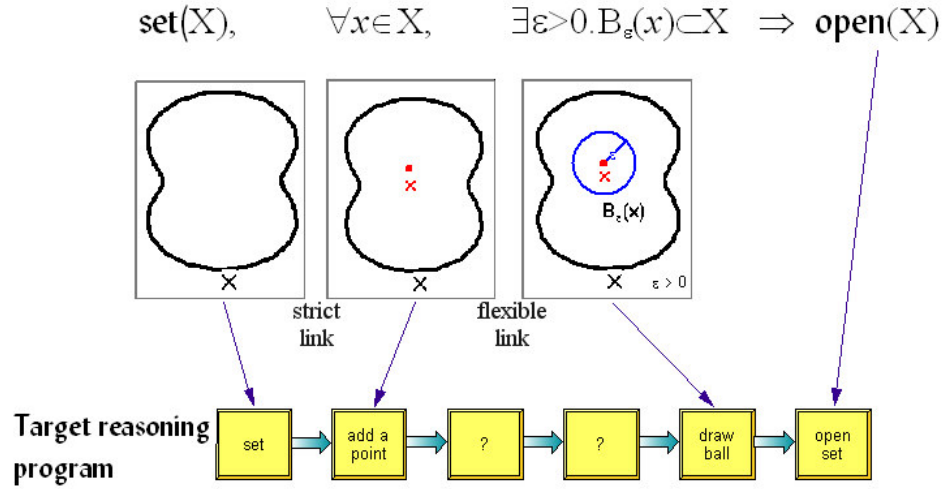


Figure 4.15. Example illustrating program matching.

Definition 4.3.3.4: Program matching II (vacuous quantification case)

Say “reasoning chain \underline{A} matches reasoning program \underline{B} with mapping m ”, where $\underline{A} = A_1 \dots A_n$, if \exists functions $m: \text{labels}(\underline{A}) \rightarrow \text{labels}(\underline{B})$, $m: \text{diagrams}(\underline{A}) \rightarrow \text{diagrams}(\underline{B})$ and $\exists j > 1$ such that:

- (a) $A_1 \dots A_j$ matches \underline{B}' a sub-program of \underline{B} with mapping m using definition 4.3.3.3
- (b) $A_{j-1} \dots A_j$ is a strict link in \underline{A}
- (c) $\exists x \in \text{objects}(A_j \setminus A_{j-1})$ such that $\text{not}(\exists m(x) \in \text{relations}(B_k))$,

where $B_k \in \text{leaves}(\underline{B})$ is an ancestor of diagram $m(A_j)$

Definition 4.3.3.5: Animated rule application

$\underline{D} \mapsto \underline{D}'$ is a valid application of rule $R: \underline{T} \mapsto \underline{T}'$, where $\underline{T}, \underline{T}'$ are reasoning chains, if:

- (a) Matching: $\exists m, m'$ such that $\underline{T} \simeq m \simeq \underline{D}$, $\underline{T}' \simeq m' \simeq \underline{D}'$ and $m = m'$ where defined.
- (b) Redraw: $\text{objects}(D_{d+1}) \subset \text{objects}(D_d)$,

$$\text{relations}(D_{d+1} \setminus D_d) = \text{relations}(T_{t+1} \setminus T_t)$$

For an example of how definition 4.3.3.4 is used, please see the proof that the empty set is open, included on the accompanying CD-ROM

Condition updating – an implementation of *strategic withdrawal*

With definition 4.3.2.4 for diagram matching, the original diagram must state all the conditions that will be required in the proof. This is of course the normal way to work. However a more flexible form of matching is possible which allows us to start without specifying conditions, and extract the correct generalisation from the reasoning used in proving the theorem. This is possible because of the link between diagrams and models, which makes it simple to test conditions as we reason. We call this way of working *condition updating*. It uses a modified version of the matching definition (given below) together with a procedure for incorporating new conditions into the reasoning.

Definition 4.3.3.6: Adaptive diagram matching

We say “diagram A matches diagram D in \underline{D} to give diagram D' in \underline{D}' ” if:

\exists a function $m: \text{labels}(A) \rightarrow \text{labels}(D)$ such that:

- (a) $\forall r(x,y) \in \text{relations}(A)$, either $r(m(x),m(y)) \in \text{relations}(D)$,
or $r(m(x),m(y))$ is true (but not observed) in D .

- (b) $\underline{D}' = \text{update}_a(\underline{D})$, $D' = \text{update}_a(D)$ as defined below by
definition 4.3.3.7.

This differs from definition 4.3.2.4 in that the observed relations in diagram A need not be observed relations in diagram D , as long as they are true. Suppose diagram A is the antecedent to a redraw rule R that we wish to use in a proof. When we find a relation $r(x,y)$ in A such that $r(m(x),m(y))$ is true (but not observed) in D then we have discovered a new condition necessary for the proof to work. We can use the rule, but it is then necessary to modify the conjecture we are working on to take the new condition $r(m(x),m(y))$ into account. This involves adding $r(m(x),m(y))$ to the initial diagram (and all diagrams in the reasoning program) – which can cause other changes to the reasoning program. We call this process *the condition update*, and it is defined as follows:

Definition 4.3.3.7: Condition updating

Suppose diagram D is part of reasoning program \underline{D} , and we wish to match diagram A to D .

\forall relations $r(x,y)$ in $\text{relations}(A)$ such that $r(m(x),m(y)) \notin \text{relations}(D)$, the relation $r(m(x),m(y))$ is added to all diagrams D_i in \underline{D}

Pruning

Suppose $m(x)$ is an object created during the proof, and therefore is not present in $\text{root}(\underline{D})$. Clearly a condition cannot be added if it concerns an object that isn't in a diagram. We deal with this by shortening the reasoning program so that the existence of object $m(x)$ becomes part of the theorem antecedent.

Let D_i be the diagram in which $m(x)$ is first drawn. Then the reasoning program is 'reverse pruned' to make D_i the root node of the 'updated' program \underline{D}' . Everything but D_i and its descendant nodes is thrown away, and D_i becomes the new, more limited, statement of the theorem antecedent. We call this *making a pruning cut at D_i* .

One effect of this is to prevent assumptions being made within only one branch of a case split (which would invalidate the case split if allowed).

Conceivably, pruning could interact with the application of an animated rule by 'cutting' the section of the reasoning program that the animated rule matched. E.g. suppose $R_a: T_1 \dots T_i \mapsto T'$ is an animated rule that was used in drawing \underline{D} with $\underline{T} \simeq n \simeq \underline{D}$, and $\text{relations}(T \setminus T_i) = \{p(a, b)\}$. If we later make a pruning cut at D_i and $n(T_1) < i \leq n(T_i)$ (i.e. if we cut the program in the middle of the pre-condition for R_a), then we can no longer apply the rule R_a to all reasoning programs \underline{D}' that match \underline{D} . We cannot envisage a scenario in which this would naturally happen, but it is a logical possibility.

In such a case, since we can no longer guarantee that rule R_a can be applied, we add its post-condition $p(a, b)$ as an extra assumption. This could entail further pruning. However since the program is of finite size, and each pruning cut reduces its size, the update process will always terminate.

When matching diagram A to diagram D , we refer to the changes made in the condition update process as update_A , and write $\underline{D}' = \text{update}_A(\underline{D})$ and $D' = \text{update}_A(D)$. Note that this notation is underspecified in that there may be several possible matchings for A , with different effects. This will not bother us though, as we will always talk of updating \underline{D} within the context of a specific matching.

We give an example of this procedure in Figure 4.16. The triangle S in diagram A has $\text{right-angled}(S)$ as an observed relation. It is allowed to match triangle T in diagram D (where this relation is true but unknown), provided \underline{D} is updated (giving reasoning program \underline{D}'). The update process adds in the $\text{right-angled}(T)$ condition to all diagrams in \underline{D} . Since the initial blank diagram of \underline{D} cannot represent $\text{right-angled}(T)$, this requires a pruning cut that removes part of the program.

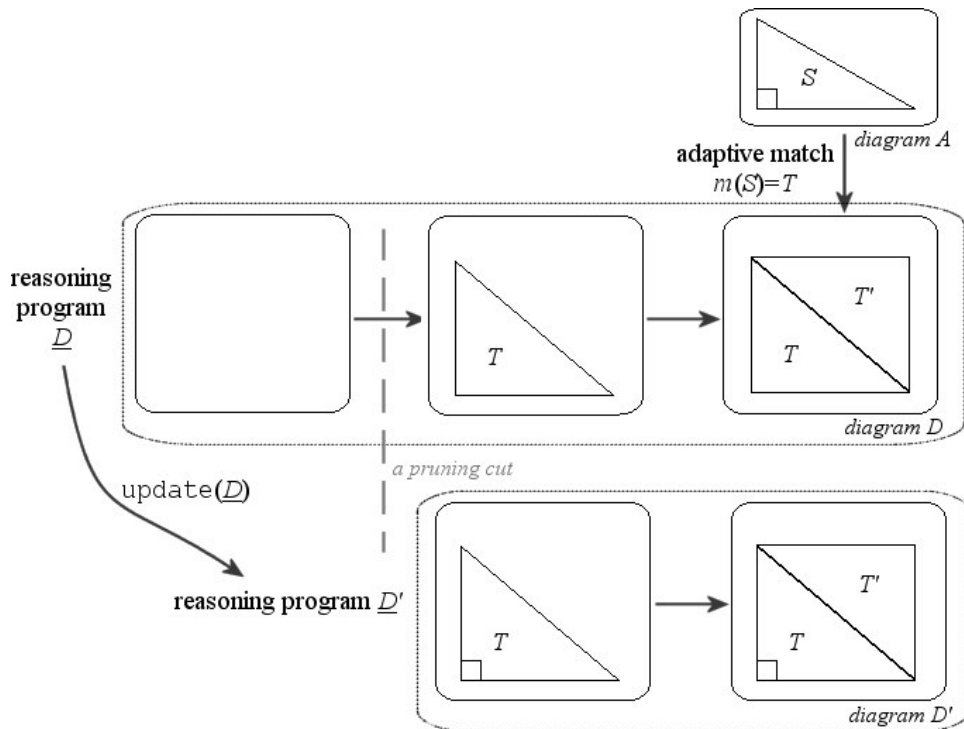


Figure 4.16. Example of adaptive matching: The right-angled condition is 'discovered' and added to the reasoning program, causing a pruning cut to remove part of the reasoning program.

Condition Updating and inaccurate diagrams / untestable relations

Condition updating makes use of the link between diagram semantics and models. This means that it requires accurate diagrams to work reliably. However it does *not* require accurate diagrams for it to be sound. Suppose we have somehow created an inaccurate diagram. Then condition updating might allow us to reach a conclusion which is false even for the model under consideration. However in doing so, it would also add extra pre-conditions to the conjecture which would guarantee – if met – that the reasoning

worked. It might turn out that these pre-conditions were impossible to meet, in which case the resulting theorem would be true, but vacuous.

To work properly, condition updating also relies on us being able to semantically test relations in the diagram. Implicit relations (such as $A \subset B$) will always be testable. However there are relations such as $\text{open}(X)$ which we cannot, in general, test. In such cases, we can still use condition updating by assuming that the condition holds. As with inaccurate diagrams, this would not lead to unsound conclusions, but could lead to a theorem that whilst seemingly meaningful is in fact vacuous.

4.3.4 Meta rules

It should be possible to devise a higher-order visual language (i.e. a language that allowed us to quantify over reasoning-programs and redraw-rules) for representing meta-rules. The representation of meta-rules in **Dr Doodle** (c.f. Figure 4.17) indicates how this might be done. However since we use only three rules which can't be formulated as normal redraw rules, we have treated them as special cases. The first of these rules is the implicit inference rule regarding specific objects, which will be dealt with in §4.3.5. The other two rules are merge and extract-rule.

Merge rule

The merge rule states that “what is true in all branches, is true generally”. It allows us to draw separate branches of a reasoning program back together.

Definition 4.3.4.1: Merge rule

Given reasoning program \underline{D} , let $\{D_i\} = \{D \in \text{leaves}(\underline{D}) : \text{false} \notin \text{relations}(D)\}$

If $\{D_i\} \neq \emptyset$, then let D' be the diagram:

$$D' = (\{x : \forall D_i, x \in \text{objects}(D_i)\}, \{r(x,y) : \forall D_i, r(x,y) \in \text{relations}(D_i)\})$$

Let \underline{D}' be the reasoning tree $\underline{D}' = \underline{D} \cup D'$ where D' is a child node for all D_i

Then $\underline{D} \mapsto \underline{D}'$ is a valid application of merge

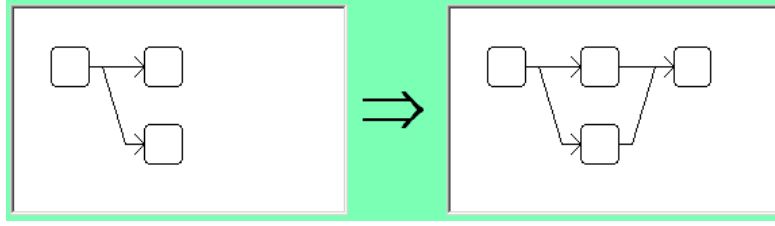


Figure 4.17. Representation of the merge rule in Dr Doodle.

Extract Rule

The extract rule allows us to create new redraw rules from reasoning programs.

Definition 4.3.4.2: Extract rule

Given reasoning program \underline{D} with root diagram D_0 ,

let $D_1...D_n = \{D \in \text{leaves}(\underline{D}) \mid \text{false} \notin \text{relations}(D)\}$ and define `extract-rule` (\underline{D}) to be the rule $R:D_0 \mapsto D_1...D_n$

4.3.5 Counterexamples

Specific objects

In §4.2.8 we discussed how diagrams can be used to present counterexamples. This involves using graphic objects that are *specific*. Specific objects are both existentially quantified and fully observed, where fully observed means that the object's parameters (e.g. the coordinates for a point, the membership formula for a set) are known and can be used in the reasoning.

Note that usually the object's precise parameters will not be important. For example, in Figure 4.13 the position of the point x (inside B and X but not Y) is important, but its precise coordinates are not. Stating the coordinates would therefore clutter up the diagram with unnecessary explicit relations. Instead we introduce a new meta-relation *specific* to state that an object is specific, but without precisely specifying which object it is. Specific objects do not generalise, and therefore anything that is true about the object drawn can be taken as true without it being proved for the general case. Hence any relations involving specific objects that are true in the diagram are added to the observed relations (i.e. the corresponding `unknown` relations are removed).

This is implemented as follows, using a special implicit inference rule. We call a physical diagram *accurate* if the drawing of the diagram objects will serve as a model for those

objects. Let \mathcal{D} be a physical diagram created from an idealised diagram D , and suppose we have a predicate `accurate` for determining when \mathcal{D} is an accurately drawn diagram (this predicate is defined in §5.2.1.2 along with a cautious algorithm for evaluating it). Then:

$$\begin{array}{l} \text{implicit}(r(x,y)) \text{ and specific}(x) \text{ and specific}(y) \\ \text{and accurate}(\mathcal{D}) \end{array} \Rightarrow r(x,y)$$

Note that only top-level existentially quantified objects can be specific. It would not make sense to talk about a “universally quantified specific point”, or a specific point $x=(1,1)$ within an arbitrary set X . Hence if an object is made specific, all the objects it depends upon must also be specific. This requirement is implemented with the pre-condition for creating specific objects that:

$$\text{specific}(x) \text{ and dependent}(x,y) \Rightarrow \text{specific}(y)$$

In principle, specific but roughly drawn objects could be used outside of counter-example reasoning. However it is not clear that they have any useful role in proving theorems of the form $\forall x.p(x) \Rightarrow q(x)$. Incorporating them into normal redraw rules would require an adjustment to the matching criterion, since meta-predicates such as `specific` and `accurate` cannot be treated like other relations (the adjustment required is to add the condition that, if $\text{specific}(x) \in \text{relations}(D)$, then $D \simeq m \simeq D'$ only if $m(x)$ and x refer to identical objects – i.e. $\mathbb{D}(m(x)) = \mathbb{D}(x)$). However, since the `specific` relation does not appear to have any clear uses outside of counter-example reasoning, we will instead restrict its use to counter-example proofs.

Counter-example proofs

A counter-example proof consists of a reasoning program \underline{D} constructed from a blank diagram D_0 using rules $\Delta \cup \{R\}$, such that $\text{false} \in \text{relations}(D)$, $\forall D \in \text{leaves}(\underline{D})$, from which we conclude that if Δ is sound then R is not sound. Note that DDL does not have mechanisms for then using this result elsewhere. That is, we do not automatically turn $\text{not}(R)$ into a lemma (c.f. §8.2.4).

4.4 Using dynamic diagram logic for analysis

We now look at adding a representation scheme to the DDL framework set out above. This representation scheme will cover a range of analysis objects and concepts, giving a diagram

logic for doing analysis proofs, which we call *DDLA*. We assume standard definitions for the basic objects in the domain.

4.4.1 Scope of DDLA

Our aim here is *not* to give a diagrammatic axiomatisation of analysis. Whilst this would be an interesting project, it goes well beyond the scope of this work. Our aim is to give a sound deduction system based in analysis which can prove an interesting range of theorems. DDLA covers the concepts of open and closed sets and monotonic and continuous functions. It does not cover some key aspects of analysis, including sequences (except in a very restricted way), differentiation and integration. In §8.2 we explore some ideas for extending this logic to cover these concepts, but these are not developed enough to be formalised yet.

4.4.2 Design issues in DDLA

The goal of DDLA is to give a formal system for reasoning in real analysis that can replace conventional algebraic approaches and is more natural. This introduces conflicting design goals between the *power* of the logic (what can be expressed and what can be proved) and its *appeal* (how easy it is to use). There are also conflicts within these goals: what is appealing for one area may not generalise well to others, plus the trade-off between simplicity⁴⁵ and flexibility which cuts across these goals.

Inevitably, these conflicts mean that the solutions we arrive at are compromises. We have already seen compromises between these goals in the set-up of the DDL framework, and we will see more here. As such, our solutions are sub-optimal for some cases (“a jack of all trades is the master of none”). It is important to remember that in designing DDLA we are not attempting to build the canonical system for analysis reasoning, since it is unlikely that such a thing is possible. Note also, that a good design for reasoning in analysis, may not be as suited to other domains (and vice-versa). Different design choices are possible, leading to different logics. Ideally our choices would be made on the basis of theory backed up by experimental results. Unfortunately we are not in a position to do that, as DDLA is the first such project in this domain.⁴⁶ Instead, we must make design choices on the basis of our own judgement, and hope that the evaluation will justify them.

⁴⁵ Note that elegant representation schemes do not necessarily correspond to elegant formalisations. Simple diagrams can sometimes require a *more* complex logic to formalise them.

⁴⁶ When exploring, you cannot consult the map.

4.4.3 Objects

DDLA covers only a subset of objects in the domain. We call these the *drawable* objects. Below we set out the different object types in DDLA, and the drawable objects for each type. Figure 4.18 gives an overview of object types and their representations.

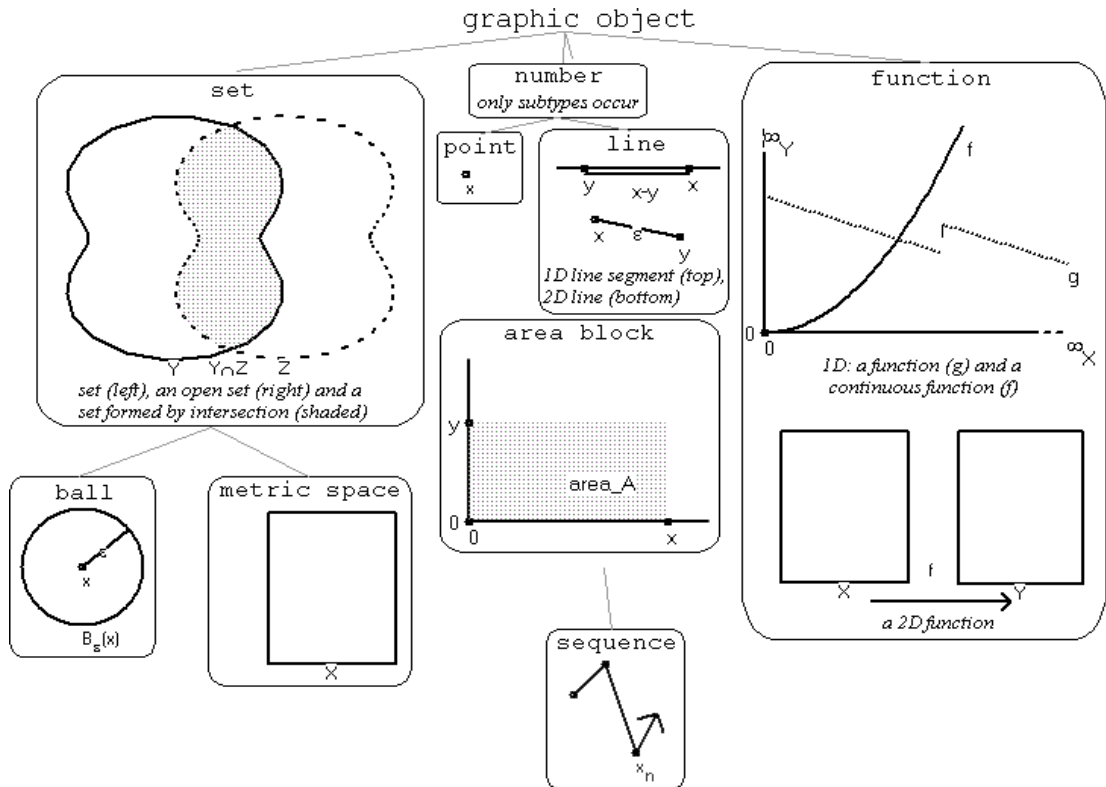


Figure 4.18. Example representations for object types in DDLA (taken from Dr Doodle screenshots).

Numbers

All numbers in \mathbb{R} are drawable, plus infinity. Numbers are a 'virtual type'; they always occur in the form of one of their subtypes.

Points

All points in \mathbb{R} and \mathbb{R}^2 are drawable.

Lines

All line segments between two drawable points are drawable. Lines do not have a direction (e.g. L from a to $b = L'$ from b to a). Lines are a subtype of Numbers, with a line segment from a to b automatically cast to the value $\|a-b\|$ within numerical statements.

Area Blocks

Area Blocks are a subtype of Numbers, equated with the value of their area.

The following basic area blocks are drawable:

- $[a,b] \times [c,d] \forall a,b,c,d \in \mathbb{R}^+$

The following rule constructively defines other drawable area blocks:

- X, Y drawable area blocks $\Rightarrow X \cap Y, X \cup Y$ drawable area blocks

Sets

Here we define several basic types of set, plus rules for composing these to create structured sets. In order for us to then reason about these sets, we have redraw-rules that work on type and structure (e.g. the set-union rule in Figure 4.26). This approach is not as powerful or flexible as the standard approach of defining sets by membership functions. However it is better-suited to the drawing-based reasoning we examine, as there is a clear link between set definitions and set drawings.

The following 'basic' sets are drawable:

- The empty set \emptyset
- Singleton sets for all drawable points
- $(a,b) \times (c,d) \forall a,b,c,d \in \mathbb{R}$
- Open 'blob' sets – amorphous shapes which loosely fit any rectangle. We require that these blobs be (technically) *centred* (also called *star-shaped*). That is given a blob set B , \exists a point $c \in B$ such that $\forall b \in B$, the line $bc \subset B$. The need for this condition is not obvious. It is added to prevent a specific type of error where a set's representation is misleading (c.f. §5.3), and does so at the cost of excluding many non-problematic sets. A more satisfying solution – though harder to formulate – would be to outlaw sets whose physical drawing can be mis-interpreted.

For each basic set, we must be able to determine whether a given point is within the set or not.

The following rules recursively define a large class⁴⁷ of drawable sets:

- X a drawable set \Rightarrow the Cauchy closure of X (i.e. X plus 'border points') is a drawable set.

⁴⁷ Note that if we allowed infinite unions of sets, we would get the Borel sets for \mathbb{R} , \mathbb{R}^2 (a common definition for measurable sets in real valued spaces).

- X a drawable set $\Rightarrow X^c$ is a drawable set (where the set complement is taken relative to the enclosing Euclidean space).
- X, Y drawable sets $\Rightarrow X \cap Y, X \cup Y$ drawable sets
- X a drawable set, f a drawable function $\Rightarrow f(X)$ is a drawable set (where defined)
- X a drawable set, f a drawable function $\Rightarrow f^{-1}(X)$ is a drawable set (where defined)

Euclidean Spaces (a subtype of Sets)

The only drawable Euclidean spaces are \mathbb{R} , line segments in \mathbb{R} , and rectangular sets in \mathbb{R}^2

Balls (a subtype of Sets)

All balls are drawable:

- $B_r(x) = \{x' : |x-x'| < r\} \quad \forall r \in \mathbb{R}, \forall x \in \mathbb{R}^2$

1D Sets (a subtype of Sets and Lines)

The following 1D sets are drawable:

- $(a, b) \quad \forall a, b \in \mathbb{R} \cup \{\infty\}$

Plus those which can be created using the construction rules for sets.

We also define a special instance of 1d-set as a constant:

- $\mathbb{R}^+ = (0, \infty)$

This constant is not necessary from a logical point of view. However it simplifies some diagrams.

Functions

Since 2D functions cannot be plotted (it would require 4 dimensional drawing), diagrams are unsuitable for exploring specific functions. Hence we only have a very limited set:

- The 'stretched identity function' f_{XY} is a drawable function $\forall X, Y$ Euclidean spaces
(this function is defined by: f_{XY} linear, f_{XY} maps X onto Y without rotation or reflection)
- The constant functions $f(x)=y$ are drawable functions \forall drawable points y
- f, g drawable functions from X to $Y, A \subset X$ a drawable set
 $\Rightarrow x \rightarrow \{f(x) \text{ if } x \in A, g(x) \text{ otherwise}\}$ is a drawable function
- $f: X \rightarrow Y, g: Y \rightarrow Z$ drawable functions $\Rightarrow x \rightarrow g(f(x))$ is a drawable function

When working in 1D the following basic functions are drawable:

- $x \rightarrow x$
- $x \rightarrow c \ \forall c \in \mathbb{R}$
- $x \rightarrow \sin(x)$

Plus those constructable using the rules:

- f a drawable function $\Rightarrow x \rightarrow 1/f(x)$ is a drawable function.
- f, g drawable functions $\Rightarrow x \rightarrow f(x)+g(x), x \rightarrow f(x).g(x), x \rightarrow f(g(x))$ are drawable functions.

This allows step functions and polynomials amongst others. It excludes functions with infinite discontinuity, such as $f(x)=\{1 \text{ if } x \in \mathbb{Q}, 0 \text{ otherwise}\}$

Note that although a function may be drawable, DDLA cannot reason about it without a redraw-rule based definition. In particular, we do not yet have such a definition for $\sin(x)$; this function is included as drawable only because it makes a good visual example.

Sequences

Because of the lack of support for natural numbers and proof-by-induction, DDL is not really equipped to reason about sequences (c.f. §8.2.5 for a discussion on how DDL could be extended to do so). We cannot define or reason about specific sequences in DDLA. However, we do introduce sequence objects which can be used to prove some theorems relating to arbitrary sequences. All sequences in \mathbb{R}^2 are drawable, but very little can be said about them. Since we will only reason about sequence properties which depend on behaviour in the limit (e.g. convergence), all relations involving sequences are defined for the sequence head (i.e. we say $r(x_n, y)$ holds if $\exists N$ such that $r(x_n \geq N, y)$ holds).

4.4.4 Constants

We require the following constants:

- $0, 1, \emptyset, \infty$
- A constant for each object type
- A constant for each of the basic functions

4.4.5 Representations

Object Representations

This describes how we draw different graphic objects. It defines bit-by-bit a *drawing function*, \mathbb{D} , that converts idealised diagrams (made up of drawable domain objects) into

physical drawings. All drawing is done with a rounding approximation of ϵ . This means objects whose representations given perfect drawing would differ by less than ϵ , have representations which cannot be distinguished. The implicit relation rules will take this into account, so that $\text{relations}(D)$ reflects that drawing is imperfect. The value of ϵ is not fixed since it depends on (a) the limitations of human vision, (b) the limitations of the drawing equipment used, and (c) the drawing scale. It should be chosen conservatively to ensure mistakes cannot be made.

Numbers

In a real valued domain, we require a representation scheme for numbers that is itself real valued. The representation should represent ordering (i.e. $a < b$), and as much as possible, it should also support the arithmetic operations: $+$, $-$, \times and \div . We consider four options here:

- 1) Use a number line, with numbers represented as points on the line, positioned according to size.

This representation is simple, intuitive and probably already familiar. It is good for representing functions too (using 2 number lines as graph axes). It represents ordering, and provides transitivity of ordering as a free ride. However it is not good for showing relations between numbers other than total ordering. For example, when representing a , b and $a+b$, $a-b$, a/b , a number line does not show the relations between these points.

- 2) Represent numbers as line segments, arranged parallel to a main number line, and positioned according to how they are created. Figure 4.19 provides an example.

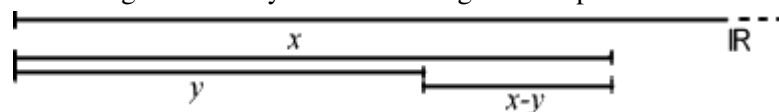


Figure 4.19. Using line segments to represent numbers.

This is also simple and reasonably intuitive. It allows addition and subtraction to be performed as graphical operations, and gives a number of free rides for these relations:

- Addition/subtraction preserve ordering, (e.g. $x > y \Rightarrow (x+z) > (y+z)$)
- Addition/subtraction as inverse operations
- Associativity (e.g. $(x+y)+z = x+(y+z)$)

It is not a suitable representation for graphing functions though, because not all line segments have a common origin.

Bringing in multiplication

The 'obvious' way to perform multiplication is by taking two lengths at right angles and considering the area of the rectangle they form. This is fine some of the time. However we also need a way of converting such rectangles into the same form as the original numbers (e.g. consider performing $2 \times 2 - 3$). For whole numbers, this is easy – the rectangle can be rearranged from a 2D matrix into a 1D strip (c.f. Jamnik's choice of representation for DIAMOND [24]). For real-valued arithmetic, it is less clear how this should be done.

3) Represent numbers produced by multiplication as *area blocks* (see Figure 4.20).

This allows multiplication to be performed graphically, and gives some free rides (e.g. order preservation: $x > y \Rightarrow x.z > y.z$ and distribution over addition: $x.(y+z) = x.y + x.z$). It allows addition and subtraction to be performed if the blocks have suitable sizes (e.g. $2 \times 2 - 1 \times 2$ is possible, $2 \times 2 - 1 \times 3$ is not). Area blocks don't support addition and subtraction in general, though, are not suitable for plotting graphs and take up more space than a number line. They also cannot represent negative numbers unless we introduce the concept of signed area, which is less intuitive.

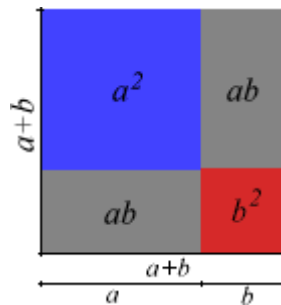


Figure 4.20. $(a+b)^2$ represented with line-segments and area blocks, giving the free inference $(a+b)^2 = a^2 + 2ab + b^2$

4) Hybrid Representation

It is possible to use a representation that switches between the above methods as appropriate, giving representations tailored to the problem under consideration. This potentially gives us the advantages of all of the above methods. However it could easily involve (if not require) several different representations of the same number at the same time, which might be confusing. Also, switching between representations might be hard to automate, in which case the user would have to perform it, which could get annoying.

Conclusion

We use a hybrid representation, combining all of the above according to the following rules:

- 1) The default option is to draw numbers as points on a horizontal or vertical number line .
- 2) Numbers are drawn as line segments (a) when two numbers (points or line segments) in the same number line are added or subtracted, or (b) when a ball is added (as the radial line).
- 3) Numbers are drawn as area blocks (a) when two positive numbers (which can be either points or line segments) on opposing graph axes are multiplied, or (b) when two area blocks are added or subtracted (as long as one of the original blocks fits inside the other – otherwise this would create negative areas). Where necessary, the user must explicitly convert area blocks into points.
- 4) If a number has two distinct representations in a diagram, one of which isn't generating any implicit relations, then this representation can be removed at the user's request.

Points

Points are drawn as small circles. Suppose x is a point, and $\mathbb{D}(x)$ is the graphic object it is drawn as, then given a rounding approximation of ε , $\mathbb{D}^{-1}(\mathbb{D}(x)) = \{x' : |x-x'| < \varepsilon\}$ (i.e. x could be read as any point within ε of its true position).

Lines

Lines are drawn as lines. A line to or from ∞ ends in ellipsis to indicate that it continues to infinity. Line segments within a number line are drawn slightly offset so that they are visible against the number line. Two lines are indistinguishable if and only if their end-points are indistinguishable. Hence a line $L=ab$ may be read as any line in $\{L' : L'=a'b', a' \in \mathbb{D}^{-1}(\mathbb{D}(a)), b' \in \mathbb{D}^{-1}(\mathbb{D}(b))\}$

Area Blocks

Area blocks are drawn as rectangles shaded transparent grey without borders.

Sets

Basic sets are represented by drawing their border line (the way in which the border is drawn depends on whether the set is open, closed, or neither/unknown). Note that for all of

the basic sets, this border will be a simple closed curve⁴⁸. Compound sets (i.e. $Z=X\cap Y$ or $Z=X\cup Y$) are represented by drawing the sets X , Y from which they are formed and shading the interior of the compound Z . Similarly, for complement sets $Z=X^c$, X is drawn, and the interior of Z is shaded. If two or more shaded sets overlap, the shading is darkened to indicate that the overlap is contained in both. There are sets in the domain for which this representation is inappropriate (e.g. highly disconnected sets). However it is a suitable representation for proving many important theorems. Sets created by the application of a function or an inverse function (i.e. $A=f(B)$ or $A=f^{-1}(B)$) are drawn as blob sets within a bounding box. This box is calculated by mapping the box of the pre-image set, then fitting a blob to that box. This representation is not generally accurate.⁴⁹ However since the idea of the blob shape is to suggest a lack of knowledge about the set's form, it should not normally be misleading.

Euclidean Spaces

Euclidean spaces are represented as rectangles. Their border represents infinity.

Balls

Balls are represented as circles, with a centre and a radial line.

1D-sets

1D-sets are represented as lines. Sets that extend to infinity are drawn ending in ellipsis (to convey continuation beyond what is drawn). This is consistent with the representation for lines. 1D-sets are assumed not to include their endpoints by default.

Functions

When working in 2D, functions are represented as arrows between Euclidean spaces. This is quite a weak representation, since whilst it shows the functions range and domain, it says nothing about what kind of function it is. Unfortunately, we would need 4 dimensions to plot a function from \mathbb{R}^2 to \mathbb{R}^2 . When working in 1D however, functions are represented by

48 A *simple closed curve* is a continuous function $f:[0,1]\rightarrow X$ such that $f(0)=f(1)$ and f is injective on $(0,1)$. i.e. a *loop*.

49 This representation is accurate for linear functions with positive determinants (i.e. stretches, translations and rotations), where the image/inverse of a blob is just the blob that fits the appropriately transformed bounding box. It is not accurate for most non-linear functions, where the blob shape may be transformed in arbitrary ways.

plotting them on graph axes. This representation is much more informative, revealing at a glance the function's behaviour up to resolution issues.

In order to reason about specific functions we need a way of accurately recognising them. This is done by adding algebraic statements (e.g. $f=\sin$). Functions without algebraic descriptions are interpreted as (stretched) identity functions. At present we do not include any reasoning that leverages the diagrammatic representations for extra power (e.g. reasoning about functions from their graphs).

Sequences

Sequences are drawn as a string of 3 consecutive points from the sequence with an arrow passing through them (indicating the order of the points). The arrow then points away, indicating that the sequence continues (see Figure 4.18).

Relation Representations

This section covers both the default interpretations that give rise to implicit relations, and the representations used for explicit relations.

Type

Object type is an implicit relation, as is object subtype. $\text{type}(x,y)$ is read as “object x is of type y ”, whilst $\text{subtype}(x,y)$ is read as “type x is a subtype of type y ”. In general, object type and object subtypes are not intuitively obvious and must be learnt. However once learnt, object type is made clear by an object's representation, and for object subtypes there are only a few relations to be learnt. The type tree is shown in Figure 4.18.

Subset

Subset is normally an implicit relation. $A \subset B$ is represented by drawing A clearly inside B . Both subset and the inside/outside relationship are partial orders (that is, they exhibit transitivity, identity and anti-symmetry), so these properties of the relation are contained in the representation. As with any implicit relation, unknown statements can be used to override the implicit relation rule above. Note that although $A \subset B$ is an implicit relation, $\text{not}(A \subset B)$ is not. That is, the fact that an object is drawn outside a given set, does not allow us to infer that it is never inside. Subset is only inferred for points, sets, sequences and numbers in line-segments.

Membership

Membership is treated in the same way as subset.

Open

Open sets are represented by drawing the border of the set with a dashed line. This signifies that the border is not part of the set. It is an old device, and may well be familiar to users. Technically, the dashes should be black and grey rather than black and white so that the border line remains a solid curve. However we will assume that the border line is clear even when drawn with a dashed line.

Closed

Closed sets are represented by drawing the set with a double border. The inside border is drawn in grey, the outside in black. Points drawn between the two borders are interpreted as being on the border of the set. Where there is not an inside border (i.e. the set has sections that are either lines or points), a double border is used (extending by necessity some way outside of the set).

Centre, radius

The centre and radius of a ball are implicit relations.

Set union and intersection

Relations such as $A=X \cup Y$ are represented implicitly.

Line end-points

The end-points of a line are represented implicitly.

Function relations and inverse relations

Relations such as $y=f(x)$ or $B=f^{-1}(A)$ are represented by an arrow from the original object (e.g. x or A) to the object created by the function application (e.g. y or B). This representation should be both intuitive and familiar. These arrows are drawn in grey, which we found made the diagrams easier to read – perhaps because it makes function relations easily distinguishable from objects. When working in 2D, these arrows are labelled with the name of the function (f , f^{-1}). When working in 1D, the arrows are drawn using the function's graph, as shown in Figure 4.21. This should also be intuitive and familiar.

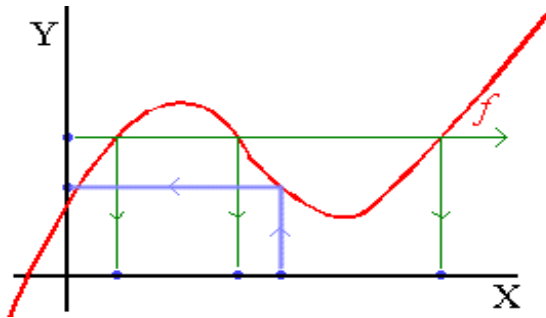


Figure 4.21. Representing function and inverse relations when working with 1D spaces.

Arithmetic relations (+, -, \times , \div)

The relation $x=y.z$ for x an area block and y, z points marking the corners or line segments marking the sides is an implicit relation. The relation $x=y+z$ for x, y, z line segments arranged in the format of Figure 4.19 is an implicit relation, and similarly for $x=y-z$. All other arithmetic relations are stated algebraically. See §4.4.5 for a discussion of this.

Equality

In 2D, equality between graphic objects is an implicit relationship. $a=b$ is represented by drawing a and b as the same object, but with two labels. The rule for observing these relations is:

If $\mathbb{D}(a) \cap \mathbb{D}(b) \neq \emptyset$, then $a=b$ is an observed relation

When two objects a, b are plotted close to each other but are not known to be equal, the statement $\text{unknown}(a=b)$ must be added to prevent this observation.

Note that although $a=b$ is an implicit relation, $\text{not}(a=b)$ is not. That is, the fact that two objects are drawn differently does not allow us to infer that they are never equal. This is slightly counter-intuitive, but means that we don't have to continually treat such cases separately.

In 1D, the same domain object can have two distinct representations, since \mathbb{R} can be drawn both horizontally and vertically to give graph axes. Where needed (i.e. where $a=b$ but $\mathbb{D}(a) \neq \mathbb{D}(b)$), equality is represented algebraically.

Ordering

Ordering (i.e. the relations $x > y$, $x < y$, $x \geq y$, $x = y$) is represented algebraically. This was probably a mistake. In most cases, ordering fits the criteria for being an implicit relation, namely that the truth or falsity of the relation is apparent from the drawing of the two objects. The exception is size ordering between line segments with different origins, where the relative sizes of the lines are not always clear. Unfortunately, the decision to represent ordering as an explicit algebraic relation is now embedded in our computer implementation and was used during our evaluation experiments. Changing this must be left as future work.

Equal-size

This is usually represented algebraically using the equals sign – which it thus overloaded in the usual way. However this only occurs between objects of different type (specifically, the three forms of number representation: area-blocks, lines and points). The clear visual difference between the different types means that this use is unambiguous. Hence a diagram $D = (\{x, L\}, \{\text{point}(x), \text{line}(L), x = L\})$ states that x and L are equal-size – *not* that x is a line and L a point. When we wish to say two objects of the same type, A, A' , are of equal size (though not equal), we represent this as $|A| = |A'|$.

Continuity

In 1D, continuity is represented graphically: continuous functions are plotted with a continuous line, whilst functions which may or may not be continuous are plotted with dashed lines. In 2D, continuity is represented algebraically (e.g. $\text{cts}(f)$).

Convergence

Convergence is represented graphically: $x_n \rightarrow x$ is represented by an arrow flowing through the points of x_n (i.e. the sequence representation) which finally points to x .

False

false is drawn as a big red cross over the rest of the diagram, indicating the situation represented is not possible.

All other relations

All other relations are represented algebraically. The notation used varies between infix (e.g. $a = b$) and prefix (e.g. $\text{cts}(f)$) as appropriate for each relation.

Quantifier Representations

Rules with mixed quantifiers are expressed using animated pre-conditions (c.f. §4.2.7). Such rules can be presented in at least two ways: using a 'TV' representation (where only one diagram is visible, and animated pre-conditions are indeed animated), or using a 'comic-strip' representation (where each diagram in the chain is shown). We anticipate that the TV representation will be easier to work with.

This still leaves open the question of how to represent the quantified objects. We could simply reinstate the algebraic symbols \forall , \exists , and represent quantifier type by labelling objects with them. However diagrams also allow other, potentially more interesting, possibilities. These include some form of drawing convention, such as colour-coding or using different shaped objects. Or - since quantifiers are introduced one at a time - quantifier type can be represented by having different transitions between frames in the animation.

Any of these representation methods would be sufficient to distinguish the two quantifier types, but they have different advantages. Using the established symbols gives the user something they may already be familiar with. Colour-coding is 'cleaner' since it does not introduce extra labelling, and this may aid comprehension.

Both of the above methods rely purely on convention for their meaning. However, since quantifier behaviour (i.e. their syntactic meaning) comes from the diagram transitions, using special transitions can attempt to convey the difference in meaning. These special transitions are animations of a different kind. They are independent of the reasoning rather than a part of it. For example, a universally quantified point could 'roam its habitat', indicating that it is not a specific point. With an existentially quantified object in a rule antecedent, the transition could indicate 'miscellaneous drawing' to illustrate that, when applying the rule, other unspecified constructions will probably be necessary at this stage. Such an approach would probably not be of interest to professional users, but could be helpful in teaching applications. However the quantifier type is not visible in the final diagram. To a certain extent, the strengths and weaknesses of these representation methods complement each other, and so they can be combined.

We use a combination of colour-coding and special transitions. Colour-coding is identical at the syntactic level to the different primitive objects used in [11], plus it can be used uniformly across types of object that are drawn in quite different ways.

Forall

Universally quantified objects are represented by drawing them in red. When playing an animation (i.e. a sequence of diagrams, either in a redraw rule antecedent or a reasoning program) they are additionally represented by a secondary animation showing a random selection of other possibilities for the object when it is first drawn. This should help indicate to the user (a) that this object stands for potentially a great many objects, and (b) which aspects of the object are arbitrary.

Exists

Existentially quantified objects are represented by drawing them in blue. When playing the animation for the antecedent to an animated redraw rule, they are additionally represented by a secondary animation showing a man thinking before drawing the object (using the odd construction line). This should help indicate to the user that there will probably be some work involved in creating this object.

4.4.6 Implicit inferences

In §4.2.8, we discussed how certain reasoning steps are 'free rides', automatically carried out in diagrams by the nature of the representation, without the need for explicit reasoning. This is captured in our formalisation by a set of implicit inference rules, which should be automatically applied by the system in any implementation of DDLA. These rules are chosen according to the criterion that any implicit representation of the antecedent must necessarily also represent the consequent:

$$X \subset Y, Y \subset Z \Rightarrow X \subset Z$$

$$X \subset X$$

$$X \subset Y, Y \subset X \Rightarrow X = Y$$

$$X = Y, Y = Z \Rightarrow X = Z$$

$$X = X$$

$$X = Y \Rightarrow Y = X$$

$$\text{line}(A), \text{line}(B), A = xy, B = xy \Rightarrow A = B$$

$$a > b, d = ac, e = bc \Rightarrow d > e \text{ (for numbers in } \mathbb{R}^+)$$

Visually obvious rules

The following rules are visually very intuitive, but not actually free-rides. It is possible – using inaccuracy – to draw diagrams such that the antecedent is implicitly represented, but the consequent is not represented at all (e.g. Figure 4.22). This is because their consequents contain explicit (though mostly graphical) relations. However any implicit representation of the antecedent will strongly indicate the consequent. Therefore, these rules should be treated like the implicit inference rules and also performed automatically:

$$X \subset Y \Rightarrow f(X) \subset f(Y)$$

$$f(X) \subset Y \Rightarrow X \subset f^{-1}(Y)$$

$$X \subset f^{-1}(Y) \Rightarrow f(X) \subset Y$$

$$x = y \Rightarrow f(x) = f(y)$$

$$x < y, y < z \Rightarrow x < z$$

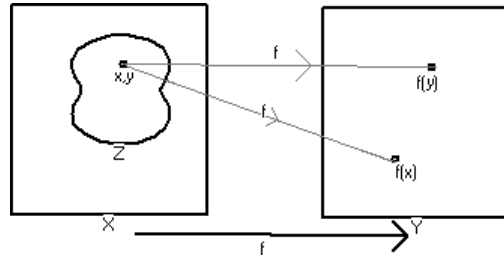


Figure 4.22. It is possible for a diagram to implicitly represent $x = y$ without stating $f(x) = f(y)$. By contrast, it is not possible to implicitly represent $x \in Z, Z \subset X$ without also representing $x \in X$.

Trivial rules

There are also reasoning rules which are neither free rides nor visually intuitive,⁵⁰ but – once learnt – are so simple that it soon feels unnatural to have to explicitly perform them. These rules should also be performed automatically, unless there is a particular reason for pointing out their application. However, this is a difficult category to define. Which rules should and should not be included in this category depends upon the user. Also, whilst a couple of applications of such rules may be trivial for most users, repeated automatic application might lead to jumps in reasoning that are not obvious to the user. We have decided that the following rules should be automated:

$$a + b = c \Rightarrow b + a = c$$

⁵⁰ Some of the rules we list here *are* visually intuitive for some combinations of number representation (c.f. the discussion of number representations in §4.4.5).

$$(a+b)+c=d \Rightarrow a+(b+c)=d$$

$$a < b \Rightarrow a+c < b+c$$

$$\text{type}(x,y), \text{subtype}(y,z) \Rightarrow \text{type}(x,z)$$

4.4.7 Emergent objects

When a set X is drawn, it might intersect with other already existing sets to create new 'emergent' sets such as $X \cap Y$, $X \cap Y^c$, etc. We experimented with having such sets created automatically as emergent objects. This involved three emergent object rules: add-set-complement, add-set-union, and add-set-intersection. However, these were found to be more hindrance than help in three ways:

- 1) The greater number of objects slows down the diagram matching algorithm in our implementation system **Dr.Doodle**.
- 2) They result in more `unknown` relations being needed, which distracts attention from more important relations. For example, consider a diagram containing sets A , B such that $B \subset A$. Then we also create the set B^c . This means that to represent $x \in A$, we must state '`unknown($x \in B^c$)`'. This is illustrated in Figure 4.23. This problem could be solved using spiders.
- 3) When applying rules, the greater number of objects leads to more valid matches, which the user has to select between. Most of these are not desirable.

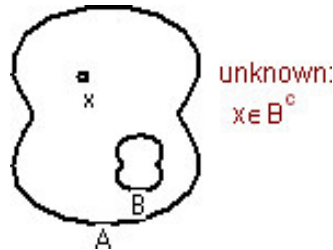


Figure 4.23. Example of using emergent sets in DDLA.

We therefore do not use emergent objects in DDLA (even though this makes our representation scheme less intuitive in places). Note that reasons (1) and (3) are related to the use of diagrams in an interactive theorem prover (or rather, to any such system based on

the user selecting and applying reasoning rules⁵¹), rather than to diagrammatic representation.

4.4.8 Rules

The complete set of rules is given in appendix A. Here we give only a small sample of the rules used in DDLA.

Function application

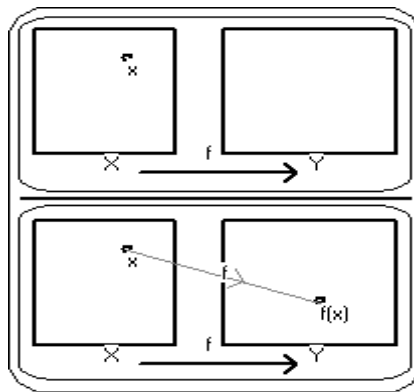


Figure 4.24. Function application for points rule.

Figure 4.24 shows a rule allowing points to be created from functions. This rule is a sort of drawing tool: it draws a new object, guaranteeing the existence of this object. For this rule, the drawing should be done automatically by any implementation. Whenever the rule is used, it is matched with a specific point and function⁵², and hence the image point can be calculated⁵³. Other drawing rules often involve user interaction to specify the desired drawing.

The interpretation of this rule is:

$\mathbb{I}(\text{Figure 4.24}) = \text{“euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X),$
 $\text{range}(f, Y), \text{point}(x), x \in X \Rightarrow \exists 'f(x)' . \text{point}('f(x)'), 'f(x)' \in Y, 'f(x)' = f(x) \text{”}$

This is trivially true from the definition of a function.

51 It might be possible to build an interface where the user interacted by making drawing changes directly to the diagram, and the system inferred which rules and matchings they were using. Such a system would not suffer from this problem.

52 Recall that in unnamed 2-dimensional functions, such as f in Figure 4.24, are taken to be the stretched identity function – c.f. §4.4.5).

53 This is true since all our functions are constructed by finite combinations of calculable base functions. It would not be true in general for functions defined using existentially quantified formulae.

Subset

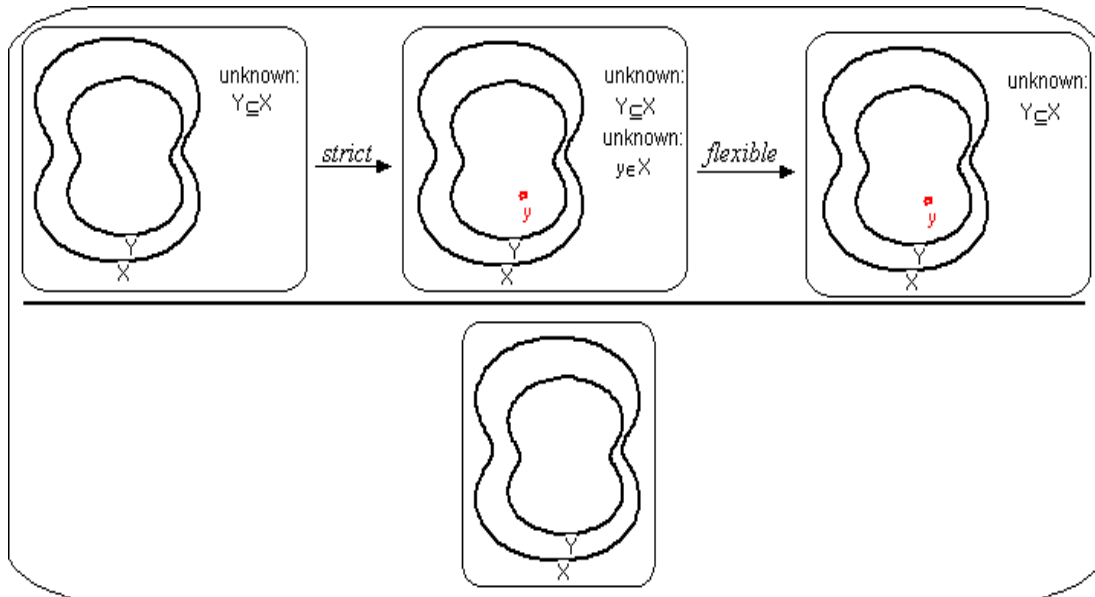


Figure 4.25. Rule for recognising $Y \subset X$.

The rule in Figure 4.25 states: Given sets X, Y , if $\forall y \in Y$, we can show $y \in X$, then $Y \subset X$. This is half of the axiomatisation for the \subset relation. The other half – $y \in Y, Y \subset X \Rightarrow y \in X$ – is handled as an implicit inference (see §4.4.6 above).

Set union

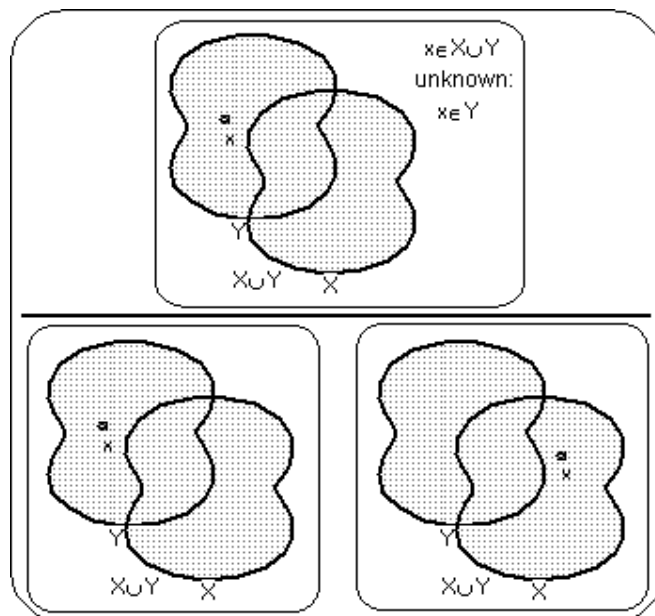


Figure 4.26. Set union definition: $x \in X \cup Y \Rightarrow x \in X \text{ or } x \in Y$

The property of a set being the union of two other sets is defined by the branch rule in Figure 4.26. The converse of this rule, “ $x \in X \Rightarrow x \in X \cup Y$ ” follows from the implicit inference rules, and so we do not need a second redraw rule.

4.5 Conclusion

This chapter has set out the core elements of the project. We have given an example of how diagrammatic reasoning can be used to prove non-trivial theorems in analysis, and presented a formal diagrammatic logic suitable for such work. We call this *Dynamic Diagram Logic* (DDL). DDL is:

- 1) Wholly diagrammatic – rules and proofs are both presented graphically
- 2) ...but heterogeneous – diagrams mix visual and sentential elements
- 3) Dynamic – reasoning is linked to the drawing process rather than interpreting a finished diagram.

DDL also introduces several new ideas. Its inference mechanisms work at a syntactic level, but leverage semantic information from the representations. This is used to simplify finding counter-examples, and to formally implement a version of Lakatos's method of strategic withdrawal, whereby a conjecture can be weakened whilst trying to prove it. Its representation scheme introduces animation as a way of representing quantifiers.

Using *implicit relations*, DDL allows facts to be inferred from the diagram, and *implicit inferences* allow some proof steps to be skipped. This provides a way of formalising reasoning from the diagram which allows for intuitive understanding.

The workings of DDL are not as simple as those of conventional logics. This should not be surprising. DDL attempts to allow for intuitive reasoning as well as being formal, and intuition is rarely simple. Also, part of the simplicity of conventional logics is due to our long familiarity with them. The early formal logics of Frege, Russell *et al* were not so simply presented.

This chapter also sets out a representation scheme for analysis concepts. We present designs for objects and relations in the domain, with some discussion of their strengths and

weaknesses. The design of these representations was guided by the aim of combining intuition, representational range and reasoning power. However in places we have had to compromise on this goal, for example over the question of existential import. The final logic (DDLA) is hopefully suitable for representing the abstract and slippery concepts of analysis.

5 Soundness

In chapter 4 we set out a method for diagrammatic reasoning (which we called *DDL*) and a logic based on this for reasoning in the domain of mathematical analysis (*DDLA*). In this chapter we evaluate the soundness of DDL and DDLA.

The usual procedure for such an evaluation would be:

- 1) Prove that the *inference mechanism* is sound. By inference mechanism, we mean the definitions in §4.3. covering rule application and the extraction of theorems from proofs.
- 2) Prove that the individual redraw rules are sound.

However, for a logic such as ours, this is not enough. The DDL inference mechanism is defined at the level of *ideal diagrams*, which are abstract diagram-descriptions – but our actual proofs and rules consist of *physical diagrams*. Hence we should also examine the link between physical and ideal diagrams. This adds a preliminary stage:

- 0) Show that physical diagrams can reliably be converted into ideal diagrams (that is, abstract diagram-descriptions). This involves examining the representation scheme set out in §4.4.

There is also another step required for those aspects of DDLA that use the diagram-model link (namely short-cuts in counter-example reasoning, c.f. §4.3.5). For these aspects to be sound, we must prove that the link between physical diagrams and models is reliable.

This chapter is organised into four sections. The first section consists of informal discussions. It examines the underlying assumptions and looks at issues relevant to proving soundness. By contrast, §5.2 consists almost entirely of definitions and results, with virtually no discussion. The structure of this section largely mirrors that of §5.1, and §5.1 serves to motivate and explain the definitions and proofs of §5.2. §5.3 then gives a detailed examination of the representation for the subset relation, which is probably the most important of the implicit relations. The final section presents some examples of showing that individual redraw rules are sound (with the complete rule-set examined in Appendix A).

5.1 Background discussions

5.1.1 Assumptions in DDLA

In setting out DDLA we have made several assumptions, which we now examine. Firstly, our work is in the domain of 'Euclidean-space analysis'. We assume that Euclidean-space analysis is a consistent and standardised theory. This allows us to prove the soundness of the individual rules of DDLA by reference to the standard theory. Our other assumptions relate to the 'preliminary stage' identified in the chapter introduction, of showing that physical diagrams can reliably be converted into abstract diagram-descriptions.

A reliable link between physical and abstract representations is necessary in any logic, diagrammatic or algebraic. For example, in conventional logics we must assume that the reasoner can reliably convert symbols on a page (the physical representation) into an array of symbols in the logic,⁵⁴ and in automated reasoning we assume that computers will behave as intended. Typically this stage is overlooked, since with sentential logics it does not raise any interesting questions. However for diagrammatic reasoning, the link between physical and logical representations is considerably more complex, and hence this stage deserves some examination.

Working in the context of an interactive theorem prover, we can rephrase this issue as follows: the computer has an internal abstract representation for the diagram using idealised graphic objects (i.e. domain objects), from which it produces the external physical representation. The user only has access to the external representation, yet must be able to find the correct interpretation. For this, we must assume:

- 1) That the computer can produce accurate physical diagrams
- 2) That the user can correctly identify graphic objects and their labels.
- 3) That the user can correctly assess observed relations.

If these assumptions hold, then the user's interpretation of the external representation will match the internal representation. These assumptions are not universally true though. In particular, assumptions 2 and 3 will fail for diagrams that are sufficiently cluttered. However if we make the deeper-level assumption of clarity – i.e. that we restrict DDLA to using clear diagrams – then they hold. This is not an unreasonable restriction.

⁵⁴ At least, we need this assumption for mathematics to be communicated.

Assumption 1

We can break this assumption down into more basic assumptions. An accurate physical diagram would be one where the observable relations correspond with the internal representation that the computer uses. To produce accurate diagrams, we require the following assumptions to hold:

- a) The surface on which diagrams are drawn obeys Euclidean plane geometry up to detectable differences.
- b) We can divide this surface into a grid such that the computer can draw domain objects on it up to the accuracy of the grid. Let $\mathbb{B} \subset \mathbb{R}^2$ be the set of squares that make up this grid. i.e. $\mathbb{B} = \{[n.\varepsilon, (n+1).\varepsilon] \times [m.\varepsilon, (m+1).\varepsilon] : n, m \in \{-N \dots N\}\}$, where ε gives the resolution of the grid and $N \in \mathbb{N}$ its size.
- c) The computer can represent all explicit relations.
- d) The computer can evaluate which relations will be implicit relations in the external representation (that is, it can correctly apply the implicit relation rules).

Assumptions (a), (b) and (c) are uncontroversial. The true geometry of physical space is unknown, but probably not Euclidean. However flat surfaces certainly appear to be Euclidean – that is, they are identical to a Euclidean surface to the naked eye – which is all we are assuming. Assumption (b) is just that we can produce bitmaps. Assumption (c) commits us to being able to represent algebraic statements, draw arrows between objects (for function and inverse relations) and draw dashed lines (open sets) and doubled lines (closed sets). It is reasonable to assume we can do all of these things.

However assumption (d) is not straightforward, since we could conceivably write an implicit relation rule that we cannot evaluate. To justify it, we must examine each of the implicit relation rules:

Type

The assumption that object type is accurately represented is subsumed by assumption 2 (discussed below). For practical purposes, consider the type representations as illustrated in Figure 4.18, §4.4.3. We require that these can be clearly distinguished. This is not problematic provided the diagram is reasonably clear (i.e. all objects are drawn large enough

for their type to be identified, and their representations are not overly obscured by other aspects of the diagram).

Subtype

Subtype is a small set of relations defining the type tree set out in Figure 4.18, §4.4.3. We take these relations to be axioms which should be learnt by users, and do not need to be represented, hence their representation cannot be inaccurate.

Subset

We examine this in §5.3 below.

Radius

The line representing the radius forms part of our representation of an open-ball. Thus if the ball representation can be correctly identified (assumption 2) then so can the radius.

Centre

If the radial line of a ball is correctly identified, then the centre point is obvious.

Equality

Implicit equality is deduced using the rule:

$$“\text{label}(A)=\text{label}(B)” \in \text{relations}(D) \text{ if } A \text{ is drawn identically to } B$$

By assumptions (a) and (b), we can calculate the bitmap representations $\mathbb{D}(A)$, $\mathbb{D}(B)$ and evaluate this relation by comparing bitmaps. Potentially, bitmap comparison also allows us to flag unclear situations (i.e. where the bitmap representations are *almost* identical) and state the negation of the relation explicitly.

Arithmetic

There are two relations that can be represented implicitly: $x=y.z$ (where x is an area-block and y, z are numbers drawn on perpendicular axes) and $x=y+z$ (where x, y, z are line-segments drawn on the same axis). By assumptions (a) and (b) we can calculate the bitmap representations $\mathbb{D}(x)$, $\mathbb{D}(y)$, $\mathbb{D}(z)$. These bitmap representations will be particularly simple (rectangles for area-blocks, small circles for points, lines for line-segments). We can then evaluate these relations reliably.

Assumption 2

Assumption 2 – that the user can correctly identify graphic objects and their labels – is not universally true. Our ability to parse a diagram depends on the clarity of the different objects. An unclear diagram can arise in several ways. Roughness of drawing (c.f. §3.1.3) can lead to diagrams where the user might be unable to correctly distinguish different objects (as shown in Figure 5.1). Ambiguity can also be caused by objects obscuring each other, or object labels obscuring other diagram details. Also, there is the possibility that a set of objects of one type might resemble objects of another type (e.g. a set of 4 lines might resemble a Euclidean space). In practice, the object types used in DDLA all have radically different representations if drawn large enough. It is therefore only in small cluttered diagrams that this problem should ever arise.



Figure 5.1. The same diagram drawn at 3 different resolutions.

Note that we only require this assumption for redraw rules – not for reasoning programs. Graphic objects created in the course of a proof can always be correctly identified regardless of the clarity of the final diagram, since the user was involved in drawing each object.

If this seems too strong, we could dispense with it by showing the drawing process whenever we present a diagram. If the user sees the diagram being drawn object by object, then the objects can reliably be identified (and matched with their labels) by examining the change at each step. Object type can be spelt out if necessary using explicit relations. So there is a reliable remedy to this potential problem. We do not insist on this remedy being used though, as it would be inconvenient when presenting a diagram to have to draw it object-by-object. A more practical variation on this (which we implement – c.f. §6.3.3) is to make our diagrams interactive, allowing the user to investigate unclear diagrams object-by-object. This protects against unclear diagrams as long as they are not actively misleading.

Assumption 3

Given assumptions 1 and 2, assumption 3 is less problematic because the implicit relation rules take into account the imprecise nature of physical representations. Assumption 3 is then equivalent to assuming that:

- a) The user properly applies the implicit relation rules.
- b) The user correctly interprets the explicit relation representations.

These postulates do require the user to learn the representational conventions used in DDLA but this is to be expected in any reasoning system. More seriously, assumption (b) is false in general. Whilst explicit algebraic statements are listed in a clear fashion, explicit graphical statements (e.g. the representation for $f(x)=y$ using arrows) can be obscured by lack of clarity in the diagram. Hence, as in assumption 2, we require the diagram to be clear. As with assumption 2, this requirement can be dropped by (a) showing the drawing process, or (b) using an interactive representation .

5.1.2 Diagram semantics (c.f. §5.2.1)

We defined a diagram to be a collection of syntactic 'graphic objects' with relations. We must now specify how the diagram semantics will work. The semantics for DDL diagrams will involve two interlinked parts:

- 1) A mapping from physical diagrams (i.e. where lines have width and some inaccuracy) to ideal diagrams (i.e. where lines are infinitely thin and anything can be accurately drawn).
- 2) A mapping from diagrams and redraw rules composed of idealised diagrams to algebra.

Our semantics will be slightly unusual in that we will fix on one underlying model: an assumed standard model for real analysis. This is not essential: we could formulate semantics in terms of 'any model'. However, fixing on one model reflects the idea running through DDLA that we can relate graphic objects in diagrams to 'domain objects'. It only makes sense to talk of such a link between diagram and domain objects in the context of a fixed model. When we talk of *models*, we really mean fragments of this one fixed model.

From physical diagrams to ideal diagrams: The drawing function

Consider two hypothetical reasoners, Alice and Bob, and suppose Alice has produced a diagram, which Bob is attempting to parse. We need to know that Bob's reading of the diagram (i.e. the objects and relations he identifies) matches what Alice intended to express. This is complicated slightly by the effects of *vagueness of representation* (one of the

implications of *roughness of drawing* identified in §3.1.3). Let D be the ideal diagram that Alice wishes to express and \mathcal{D} the physical diagram she draws. Due to vagueness of representation, Bob cannot, in general, deduce D from \mathcal{D} , since the precise details of the objects in D have been slightly blurred. For example, it might not be possible to distinguish between a point drawn at x - y coordinates $(1.5, 1.5)$ and one drawn at $(1.52, 1.49)$. However, this does not matter, as long as Bob's parsing of \mathcal{D} is not significantly different from D . Let D' be the ideal diagram that Bob thinks \mathcal{D} expresses. The key thing is, that although D' might not equal D , whatever relations Alice wishes to express about D must also hold for D' .

To state this formally, we define the drawing process as a function \mathbb{D} (the *drawing function*) that creates physical diagrams from ideal diagrams. We can then express the requirement that the drawing process must not obscure important details as a condition on \mathbb{D} :

$$\forall \text{ ideal diagrams } D, D' \text{ such that } \mathbb{D}(D) = \mathbb{D}(D'), \text{ relations}(D) = \text{relations}(D')$$

Bitmaps

Since we are using computers, our diagram objects will be drawn as *bitmaps*. A bitmap is a physical grid of coloured squares (pixels). This has a physical resolution, which is the smallest distance we can distinguish (the size of a pixel). In §5.3 we will assume that the physical resolution is sufficiently big for users to distinguish individual pixels. The grid has a finite size. However by using zooming/scrolling, we can allow this to be arbitrarily large. We associate this physical grid with an abstract grid laid over \mathbb{R}^2 . This implies specifying an orientation and an abstract resolution (the width of a pixel in \mathbb{R}^2). The abstract resolution can be changed during reasoning by zooming.

From ideal diagrams to algebra: The interpretation function

When converting diagrams and diagrammatic rules into algebra, we wish to discard the graphical information which (almost) specifies each object and keep only the relational information. This work is largely performed by the implicit relation rules. That is, the function $\text{relations}(D)$ performs most of the task of diagram interpretation. However, the relations function does not handle quantification, so we shall need to define an interpretation function for this aspect.

Note that except for the relation *specific*, all relations in DDLA are valid statements in a conventional axiomatisation of the domain. *specific* is an instruction to view a graphic object as representing only the object drawn. We will not give it a semantics directly. Instead, its meaning comes from the implicit relation rule that uses it (c.f. §4.3.5).

Consistency, accuracy and soundness (c.f. §5.2.1.2, §5.2.1.5)

Some diagrammatic proofs, such as the proof of Pythagoras's theorem (Figure 1.1, §1), can be formalised in terms of actions that preserve area. This idea is not applicable to DDLA, where many of our actions consist of creating new objects, and we are not generally interested in area. Instead, we use the more abstract idea from sentential logic of actions that preserve consistency (formalised by defining the concept of a *consistent* diagram, which is a diagram representing a situation that is possible).

Given the concept of a consistent diagram, we can then define what a sound redraw rule is: A sound rule is one that preserves consistency.

We also need a concept to cover when the diagram-model link is trustworthy. We introduce the term *accurate*, to describe a representation that is itself an example of the situation it describes. That is, an accurate diagram is a model for the relations it expresses.

Note that consistency does not imply accuracy. DDLA allows consistent diagrams to be inaccurate. This could hinder the intuitive understanding that diagrams give. Consistent but inaccurate diagrams should be avoidable, since if a diagram is consistent, then it has a model - and drawing that model would give an accurate diagram. So we *should* only use inaccurate diagrams when representing inconsistent situations. There are several reasons, however, why we do not make it a requirement that consistent diagrams must also be accurate:

- 1) The question of whether or not a diagram is consistent is undecidable,⁵⁵ so we would not know *when* to enforce such a requirement.
- 2) The question of whether or not a diagram is accurate does not have a decision procedure,⁵⁶ hence we do not know *how* to enforce such a requirement.

⁵⁵ This is roughly equivalent to the decidability of first order predicate logic with quantifiers.

⁵⁶ This is because there are not decision procedures for testing relations such as *open(X)*. This problem might be undecidable.

- 3) We might know that an accurate diagram exists for the situation we wish to represent, but be unable to find it. We can nevertheless perform valid reasoning about such a situation using *inaccurate* diagrams.

Evaluating accuracy

Accuracy is used in conjunction with the `specific` relation to allow some reasoning short-cuts. Fortunately, when evaluating accuracy, we do not need to worry about false negatives, since a false negative could only block a valid short-cut, rather than lead to a mistake (c.f. §4.3.5 where we define specific objects, or §5.2.2 where we examine the usage of accuracy with specific objects). Therefore we can take a precautionary approach to evaluating diagram accuracy, assuming objects to be inaccurate unless we are certain otherwise. The procedure we will use (set out in Definition 5.2.1.5) is quite crude, and we hope that it can be extended (c.f. §8.2.3). It fails to recognise accuracy in a large number of accurate diagrams, which limits where we can currently use diagrammatic shortcuts in counter-example proofs.

Implicit inference rules and emergent object rules

Except for the specific-object rule (c.f. §4.3.5), implicit inference rules and emergent object rules can be replaced with equivalent simple redraw rules. The difference is only that they are applied automatically. Therefore, for the purposes of showing soundness, we can treat them as simple redraw rules (this reduces the number of proofs we have to give for soundness-of-inference). The specific-object inference rule is different because it uses meta-predicates to draw on the semantic properties of the diagram. We treat it separately in §5.2.2.

5.1.3 From soundness in DDLA to soundness in standard analysis

Although unlikely, a rule could potentially be sound in DDLA whilst its interpretation is false in standard analysis. This is because DDLA can only reason about a subset of the objects in the domain (the drawable objects). Hence a rule could be sound for all *drawable* cases, but not for all cases. For example, suppose we could state “All functions are piecewise continuous” as a DDLA rule. This rule would be sound for DDLA (which does not define any infinitely discontinuous functions), but its algebraic interpretation would be

unsound. So soundness in DDLA is a *strictly weaker* condition than soundness in standard analysis. To make the full range of domain objects drawable, we would require:

- 1) Representations for objects in \mathbb{R}^n for any n .
- 2) Infinite closure of the rules that generate complex objects from basic objects, as opposed to finite closure.

Neither of these is realistic without using iconic (and largely algebraic) representations a lot more heavily (something that is worth investigating, but would require considerable design and development work). However we can consider the hypothetical reasoning system where they hold, and all objects in the domain are drawable. We call this logic 'Extended DDLA' or *EDDLA*.

As we show in Theorem 17, soundness in EDDLA is equivalent to soundness in standard analysis. Also, since EDDLA includes DDLA, soundness in EDDLA is a stronger criterion (i.e. soundness in EDDLA implies soundness in DDLA). We therefore work in EDDLA when showing that individual redraw rules are sound. Note that the inclusion of extra domain objects within the representation scheme does not affect the soundness or otherwise of the inference mechanisms. This is because the inference mechanisms are defined at the more abstract level of DDL, without reference to the objects covered. Hence the results given in §5.2.2 regarding the inference mechanism apply equally to DDLA, EDDLA and any other logic built using the DDL inference framework.

5.2 Formal definitions and results

This section uses the notational conventions in §4.3.1 and the definitions from §4.3.

5.2.1 Diagram semantics (c.f. §5.1.2)

Let \mathbb{A} be the conventional axiomatisation of Euclidean-space analysis, and \mathbb{M} a model for \mathbb{A} that contains \mathbb{R}^2 . We assume that such an \mathbb{M} exists. Up till now, we have talked of diagrams consisting of *domain objects* and *relations*. We can now specify that domain objects are objects in \mathbb{M} . Given an ideal diagram D , a *substitution* for D is a function $\sigma: \text{objects}(D) \rightarrow \text{objects}(\mathbb{M})$.

Our diagrams are produced by colouring squares in a grid. Let \mathbb{B} be the set of squares that we draw on. $\mathbb{B} = \{[n.\varepsilon, (n+1).\varepsilon] \times [m.\varepsilon, (m+1).\varepsilon] \mid n, m \in \{-N \dots N\}\}$, where ε gives the resolution of the grid and N its size. We associate \mathbb{B} with a physical grid of squares. Note that this implies specifying an orientation and a drawing scale (i.e. a conversion between real numbers and physical distances).

The drawing function \mathbb{D} (c.f. §5.1.2)

We assume that all physical diagrams are created from ideal diagrams as using a *drawing function* \mathbb{D} , that maps drawable domain objects onto physical graphic objects (given a drawing surface \mathbb{B}). Given a drawable domain object x , §4.4 specifies how to represent x as a set y in \mathbb{R}^2 . We then need to convert y from \mathbb{R}^2 to the drawing surface. The drawing function we use is a bitmap projection of the object representations specified in §4.4, whereby we colour an entire grid square if any part of it is coloured by the ideal representation of the object.

Definition 5.2.1.1: The drawing function \mathbb{D}

Define $\mathbb{D}: \{\text{drawable objects}\} \rightarrow \mathbb{B}$ by $\mathbb{D}(x) = \mathbb{D}(y) = \{B \in \mathbb{B} : B \cap y \neq \emptyset\}$, where y is the set in \mathbb{R}^2 representing x as specified in §4.4 (e.g. for points, $y = \{x\}$, for sets, $y = \text{border}(x)$).

For convenience, we overload this function as follows: Let $Z = \bigcup_{B \in \mathbb{D}(x)} B$. Then we define $\mathbb{D}: \{\text{drawable objects}\} \rightarrow \mathbb{R}^2$ by $\mathbb{D}(x) = Z$.

Now given an ideal diagram D , we define

$$\mathbb{D}(D) = (\{\mathbb{D}(x) : x \in \text{objects}(D)\}, \{r : r \in \text{relations}(D)\}).$$

Note that this definition draws on Assumption 1.d (that the user can determine which relations are implicit) and Assumption 1.c (that the computer can create representations for all necessary explicit relations) from §5.1.1.

We can also define the inverse drawing function \mathbb{D}^{-1} . Since \mathbb{D} is many-to-one, \mathbb{D}^{-1} will map physical diagrams to *sets* of ideal diagrams.

The interpretation function \mathbb{I} (c.f. §5.1.2)

Definition 5.2.1.2: The interpretation function \mathbb{I}

Given an ideal diagram D , the *interpretation function* \mathbb{I} converts D into algebra as follows: $\mathbb{I}(D) = \{r(x,y) \in \text{relations}(D) \mid r \neq \text{specific}\}$

Given physical diagram \mathcal{D} , let $\mathbb{I}(\mathcal{D}) = \mathbb{I}(D)$, where D is any ideal diagram in $\mathbb{D}^{-1}(\mathcal{D})$.

This is well-defined given Assumption 3 (c.f. §5.1.1), which implies that \forall ideal diagrams D, D' such that D, D' are represented by \mathcal{D} , we have $\text{relations}(D) = \text{relations}(D')$.

Given a diagram D in reasoning program \underline{D} , define $\mathbb{I}(D|\underline{D}) = Q(D|\underline{D}).\mathbb{I}(D)$, using the quantification interpreter function $Q(D|\underline{D})$ as defined in Definition 4.3.2.3.

Given a branch redraw rule $R: D_0 \mapsto D_1 \dots D_n$ such that $Q(R) = \text{"}\exists\text{"}$,

let $\mathbb{I}(R) = \mathbb{I}(D_0) \Rightarrow \exists \cup_i \text{labels}(D_i) \setminus \text{labels}(D_0) . \mathbb{I}(D_1) \vee \dots \vee \mathbb{I}(D_n)$

Given a branch redraw rule $R: D_0 \mapsto D_1 \dots D_n$ such that $Q(R) = \text{"}\forall\text{"}$,

let $\mathbb{I}(R) = \mathbb{I}(D_0) \Rightarrow \forall \cup_i \text{labels}(D_i) \setminus \text{labels}(D_0) . \mathbb{I}(D_1) \vee \dots \vee \mathbb{I}(D_n)$

Note that if diagram A matches diagram B , then $\mathbb{I}(A)$ will unify with $\mathbb{I}(B)$.

Note that universally quantified rules are interpreted as tautologies. They are useful only in conjunction with animated rules.

Consistent and Accurate Diagrams (c.f. §5.1.2)

Definition 5.2.1.3: Consistency

Given a diagram D in reasoning program \underline{D} , and substitution σ , we say D is *consistent under* σ if:

$$\mathbb{M} \models \sigma(\mathbb{I}(D|\underline{D}))$$

We say D is *consistent* if there exists such a σ (i.e. there are objects in \mathbb{M} for which the situation represented by D is true).

Given a reasoning program \underline{D} , we say \underline{D} is consistent if \exists diagram

$D \in \text{leaves}(\underline{D})$ such that D is a consistent diagram.

Definition 5.2.1.4: Accurate diagrams

Given an ideal diagram D in reasoning program \underline{D} , let

$\sigma_I: \text{objects}(D) \rightarrow \text{objects}(\mathbb{M})$ be the substitution that maps objects in D onto themselves.

Then we say D is *accurate* if $\mathbb{M} \models \sigma_I(\mathbb{I}(D))$

(i.e. the situation represented by D is true for D itself, hence D can act as a model).

We call a physical diagram \mathcal{D} accurate if $\exists D \in \mathbb{D}^{-1}(\mathcal{D})$ such that D is accurate.

Evaluating accurate(D)

In this project we use the following definition for evaluating accuracy.

Definition 5.2.1.5: Evaluating accurate(D)

If $\forall r \in \text{relations}(D)$, r is implicit in D and r is one of $\{\text{type}, \text{subtype}, \subset, \in, \text{centre}, \text{radius}\}$ then $\text{accurate}(D)$ is true.

As discussed in §5.1.2, this evaluation scheme ignores a great number of accurate diagrams. Implicit equational relations and relations of a 'functional nature' (e.g. $a=b$ or $y=f(x)$) are not suitable for inclusion in accurate diagrams, due to the possibility of exploiting the tiny (but potentially important) differences between objects which are indistinguishable when drawn (due to vagueness of representation – c.f. §3.1.3). We now show that (without the closed world assumption) it is sound:

Theorem 1: Definition 5.2.1.5 gives no false positives

Proof

Given a physical diagram \mathcal{D} , we need to show that if $\text{accurate}(\mathcal{D})$ evaluates to true using Definition 5.2.1.5, then \exists domain objects $X_1 \dots X_m$ such that

$\mathbb{D}(\{X_1 \dots X_m\}) = \text{objects}(\mathcal{D})$, and $\{X_1 \dots X_m\} \models \text{relations}(\mathcal{D})$.

Note that $\mathbb{D}^{-1}(\mathcal{D}) \neq \emptyset$ and let D be the ideal diagram from which it was created (recall that this must exist, but cannot be uniquely identified – c.f. §5.1.2). It is sufficient to show that $\text{objects}(D) \models \text{relations}(\mathcal{D})$. This requires an examination of the relations allowed by Definition 5.2.1.5 and the implicit inference rules that generate them.

type: These relations are copied straight from D to \mathcal{D} . It is an assumption of DDLA that the user can reliably extract them, which implies that

$$\text{type}(D) = \text{type}(D') \quad \forall D' \in \mathbb{D}^{-1}(\mathcal{D})$$

subtype: These relations are true for all diagrams.

subset: c.f. Theorem 24 where we show that for diagrams without functions,

$$\text{implicit}(A \subset B) \Rightarrow A' \subset B' \quad \forall A', B' \in \mathbb{D}^{-1}(\mathcal{D})$$

radius and centre: Suppose $B_r(x)$ is a ball in \mathcal{D} . Let $B'_{r'}(x')$ be the ideal ball from which it was drawn. Then the radius and centre relations are true for objects B', r', x' .

These are all the relations allowed in \mathcal{D} by Definition 5.2.1.5. Thus if we evaluate $\text{accurate}(\mathcal{D})$ to true, then $\text{relations}(\mathcal{D})$ will be true for $\text{objects}(D)$ as required.

Sound rules

A redraw rule is sound if it preserves the consistency of any reasoning program that it is applied to:

Definition 5.2.1.6: Soundness for branch rules

A branch rule R is *sound* if \forall diagrams $D_0 \dots D_n$ such that $D_0 \mapsto D_1 \dots D_n$ is a valid application of R , and \forall substitutions σ such that D_0 is consistent under σ , then $\exists i \in \{1..n\}$ such that D_i is consistent under substitution σ .

Definition 5.2.1.7: Soundness for animated rules

An animated rule R is *sound* if \forall reasoning programs $\underline{D}, \underline{D}'$ such that \underline{D} was drawn using sound redraw rules Δ and $\underline{D} \mapsto \underline{D}'$ is a valid application of R , then $\text{root}(\underline{D})$ consistent under substitution $\sigma \Rightarrow \underline{D}'$ is consistent under substitution σ .

5.2.2 Soundness of the inference mechanism

We now present results to show that DDL is a sound logic. That is, that the inference mechanisms defined in §4.3 are sound according to the definitions given in §5.2.1 (that is, that they preserve consistency). We break this up into the following sections:

- 1) The 'basic' inference mechanism, comprising simple, branch and animated redraw rules with a generalisation procedure that creates new redraw rules from reasoning programs.
- 2) Adding adaptive matching and condition updating to the basic inference mechanism.
- 3) Using the meta rules on finished proof programs.

4) Counter-example proofs.

We then show that sound redraw rules convert into sound algebraic theorems, providing soundness.

The basic inference mechanism

The results in this section culminate in the theorem:

Given \underline{D} , a reasoning program drawn using redraw rules Δ , let N be the new redraw rule $N:\text{root}(\underline{D}) \mapsto \text{leaves}(\underline{D})$. If Δ are sound rules, then N is a sound rule.

This means that redraw rules can be chained together to give a program for redrawing diagrams which:

- 1) Can be applied to any diagram that matches $\text{root}(\underline{D})$. That is, if $\text{root}(\underline{D})$ matches diagram A , then the sequence of redrawings used on $\text{root}(\underline{D})$ can also be used on A .
- 2) Can be summed up by the example transformation $\text{root}(\underline{D}) \mapsto \text{leaves}(\underline{D})$.
- 3) Preserves consistency when applied to other diagrams.

Our proof will use induction over reasoning program structure. Lemma 4/Corollary 4.1 give the base case for (1) above: that a reasoning program consisting of just one rule R applied to a diagram D , can also be applied to any diagram D matches. Lemma 6 then covers the step case: that a reasoning program \underline{D} can be applied to any diagram A that matches $\text{root}(\underline{D})$. Finally, to demonstrate that a redraw rule N created from a reasoning program is sound, we examine its application to an arbitrary diagram A , and show that such applications preserve consistency.

The following results use Definition 4.3.2.4 (normal 'unadaptive' matching) as the basis for diagram matching. Adaptive matching will be dealt with in lemmas 8-10. For matching between reasoning programs, we treat non-vacuous and vacuous matching (Definitions 4.3.3.3 and 4.3.3.4 respectively) as separate cases.

Lemma 2: Diagram matching is a transitive relation.

Proof:

Trivial.

Lemma 3 (Simple rule application is transitive): Given simple rules $R: T_1 \hookrightarrow T_2$, $N: D_1 \hookrightarrow D_2$ such that $D_1 \xrightarrow{R} D_2$, and diagrams A_1 , A_2 such that $A_1 \xrightarrow{N} A_2$, then $A_1 \xrightarrow{R} A_2$

Proof:

There are 3 conditions to check in Definition 4.3.2.4. They all follow easily because they are themselves transitive:

a) Matching: $T_i \simeq n_i.m_i \simeq A_i$ by Lemma 2.

$$m_1 = m_2 \text{ where defined, } n_1 = n_2 \text{ where defined} \Rightarrow n_1.m_1 = n_2.m_2 \text{ where defined.}$$

b) Object bijectivity: $m_2(\text{objects}(T_2 \setminus T_1)) = \text{objects}(D_2 \setminus D_1)$, $n_2(\text{objects}(D_2 \setminus D_1)) = \text{objects}(A_2 \setminus A_1)$ so $n_2.m_2(\text{objects}(T_2 \setminus T_1)) = \text{objects}(A_2 \setminus A_1)$.

$$\text{Similarly } n_1.m_1(\text{objects}(T_1 \setminus T_2)) = \text{objects}(A_1 \setminus A_2).$$

c) Relation bijectivity: $r(t, t') \in \text{relations}(T_2 \setminus T_1) \Leftrightarrow r(m_2(t), m_2(t')) \in \text{relations}(D_2 \setminus D_1) \Leftrightarrow r(n_2.m_2(t), n_2.m_2(t')) \in \text{relations}(A_2 \setminus A_1)$

$$\text{Similarly, } r(t, t') \in \text{relations}(T_1 \setminus T_2) \Leftrightarrow r(n_1.m_1(t), n_1.m_1(t')) \in \text{relations}(A_1 \setminus A_2)$$

Lemma 4: Given simple redraw rules $R: T_1 \hookrightarrow T_2$, $N: D_1 \hookrightarrow D_2$ with $D_1 \xrightarrow{R} D_2$. Let $T_i \simeq m_i \simeq D_i$, and suppose A_1 , A_2 are diagrams such that $D_1 \simeq n_1 \simeq A_1$, then if $A_1 \xrightarrow{R} A_2$ with mapping $T_1 \simeq n_1.m_1 \simeq A_1$, we also have $A_1 \xrightarrow{N} A_2$

Proof:

We define a matching $D_2 \simeq n_2 \simeq A_2$ from the existing matchings by:

$$n_2(x) = \begin{cases} n_1(x) & \text{if } x \in \text{labels}(D_1) \\ m.m_2^{-1}(x) & \text{if } x \in \text{labels}(D_2 \setminus D_1) \end{cases}$$

Figure 5.2 illustrates how the various mappings fit together.

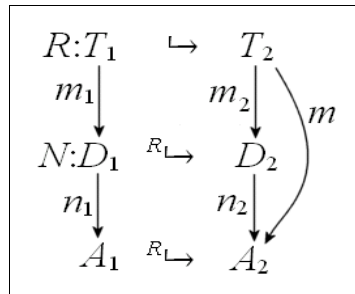


Figure 5.2. Diagrams and mappings used in Lemma 4.

We need to show that n_2 is a valid matching according to Definition 4.3.2.4.

Clearly it is a well-defined function from D_2 to A_2 .

We also require: $\forall r(x,y) \in \text{relations}(D_2), r(n_2(x), n_2(y)) \in \text{relations}(A_2)$

Suppose $r(x,y) \in \text{relations}(D_1)$

Then $r(x,y)$ is unaffected by the application of R .

$\Rightarrow r(n_2(x), n_2(y)) = r(n_1(x), n_1(y)) \in \text{relations}(A_1)$

$r(x,y)$ is unaffected by the application of R under matching m_1

$\Rightarrow r(n_1(x), n_1(y))$ is unaffected by the application of R under matching $n_1.m_1$

$\therefore r(n_1(x), n_1(y)) \in \text{relations}(A_2)$

Otherwise $r(x,y) \in \text{relations}(D_2 \setminus D_1)$

$\Rightarrow r(m_2^{-1}(x), m_2^{-1}(y)) \in \text{relations}(T_2 \setminus T_1)$

since $\text{changes}(D_2, D_1) = \text{changes}(T_2, T_1)$

$\Rightarrow r(m.m_2^{-1}(x), m.m_2^{-1}(y)) \in \text{relations}(A_2 \setminus A_1)$

since $\text{changes}(A_2, A_1) = \text{changes}(T_2, T_1)$

hence $r(n_2(x), n_2(y)) \in \text{relations}(A_2)$

Now $n_2 = n_1$ where defined by definition. Also $\text{changes}(A_2, A_1) = \text{changes}(T_2, T_1)$

and $\text{changes}(D_2, D_1) = \text{changes}(T_2, T_1)$ so $\text{changes}(A_2, A_1) = \text{changes}(D_2, D_1)$.

This fulfils all the conditions for redrawing, so $A_1 \mapsto A_2$ is a valid application of rule N .

Corollary 4.1: Given branch redraw rules $R: T_0 \mapsto T_1 \dots T_k$, $N: D_0 \mapsto D_1 \dots D_k$ such that $D_0 \xrightarrow{R} D_1 \dots D_k$ with matching m , and suppose $A_0 \dots A_k$ are diagrams such that $D_0 \simeq n_0 \simeq A_0$ and $A_0 \xrightarrow{R} A_1 \dots A_k$ with matching $T_0 \simeq n_0.m_0 \simeq A_0$, then we also have $A_0 \xrightarrow{M} A_1 \dots A_k$

Follows by applying Lemma 4 to each branch.

Lemma 5 (Transitivity of program matching): Given reasoning programs \underline{A} , \underline{B} , \underline{C} such that $\underline{A} \simeq m \simeq \underline{B}$, $\underline{B} \simeq n \simeq \underline{C}$ using Definition 4.3.3.3, if \underline{B} , \underline{C} are constructed programs then $\underline{A} \simeq n.m \simeq \underline{C}$

Proof:

Given Lemma 2 and Lemma 3, Definition 4.3.3.3 is transitive provided that when $A-A'$ is a strict link in \underline{A} , we have $m(A)-m(A')$ a strict link in \underline{B} . This is trivially true, since all links in constructed programs are taken to be strict.

Lemma 6: Suppose diagram D_0 is redrawn to give program \underline{D} using redraw rules Δ , and A_0 is any diagram such that $D_0 \simeq A_0$. Then A_0 can be redrawn using Δ to give reasoning program \underline{A} such that $\underline{D} \simeq \underline{A}$ with Definition 4.3.3.3.

Proof by induction on $|\text{rules}(\underline{D})|$

If $|\text{rules}(\underline{D})|=1$ then true by Corollary 4.1.

Suppose $|\text{rules}(\underline{D})|=K>1$

Let $r \in \text{rules}(\underline{D})$ be the final rule used to draw \underline{D} , let \underline{D}^{-1} be the reasoning program before R was applied, and let $D^{-1} = \text{target}(R, \underline{D}^{-1})$ be the diagram R was applied to.

By the inductive hypothesis, A_0 can be redrawn to give \underline{A}^{-1} such that $\underline{D}^{-1} \simeq n \simeq \underline{A}^{-1}$ with Definition 4.3.3.3. Let A^{-1} be the diagram $n(D^{-1})$.

Case 1: R is a simple/branch rule, $R: T_0 \mapsto T_1 \dots T_n$

By Lemma 2, T_0 matches A^{-1} , so we can apply R to A^{-1} to create \underline{A} , and using Corollary 4.1, we have $\underline{D} \simeq \underline{A}$

Case 2: R is an animated rule, $R: T_1 \dots T_n \mapsto T'$

Case 2a: $T_1 \dots T_n \simeq m \simeq \underline{D}^{-1}$ using Definition 4.3.3.3 (normal matching)

By Lemma 5, $T_1 \dots T_n \simeq n.m \simeq \underline{A}^{-1}$, so we can apply R to \underline{A}^{-1} to create \underline{A} , and using Corollary 4.1, we have $\underline{D} \simeq \underline{A}$

Case 2b: $T_1 \dots T_n \simeq m \simeq \underline{D}^{-1}$ using Definition 4.3.3.4 (vacuous quantification matching)

Let $T_1 \dots T_j$ be the fragment of \underline{T} that matches \underline{D}^{-1} , and $x \in \text{objects}(T_j \setminus T_{j-1})$ such that $\text{not}(\exists m(x)) \in \text{relations}(D^{-1})$.

Then by Lemma 5, $T_1 \dots T_j$ matches \underline{A}^{-1} with Definition 4.3.3.3.

Also, since $D^{-1} \simeq A^{-1}$, we have $\text{not}(\exists n.m(x)) \in \text{relations}(A^{-1})$. Therefore $\underline{T} \simeq n.m \simeq \underline{A}^{-1}$ with Definition 4.3.3.4. Hence we can apply R to \underline{A}^{-1} create \underline{A} , and using Corollary 4.1, we have $\underline{D} \simeq \underline{A}$

□

Theorem 7: Suppose D_0 is redrawn to give \underline{D} using redraw rules Δ . Let N be the rule $N:D_0 \mapsto \text{leaves}(\underline{D})$. If Δ are sound rules, then N is a sound rule.

Proof:

Let $A_0 \dots A_k$ be diagrams such that $A_0 \xrightarrow{N} A_1 \dots A_k$ using matching n . If A_0 is inconsistent then we have nothing to prove, so suppose A_0 is consistent with substitution σ .

$D_0 \simeq A_0$, so by Lemma 6, we can redraw A_0 using Δ to give reasoning program \underline{A}' such that $\underline{D} \simeq n' \simeq \underline{A}'$. Moreover, we can choose labels for the new objects in \underline{A}' such that $n' = n$.

Now Δ sound $\Rightarrow \underline{A}'$ is consistent with substitution σ , since sound rules preserve consistency by definition. Therefore, \exists consistent diagram $A' \in \text{leaves}(\underline{A}')$

Let $A \in \{A_1 \dots A_k\}$ be the diagram such that $n'.n^{-1}(A) = A'$.

Now $\text{changes}(A_0, A) = \text{changes}(A_0, A')$, so $\text{relations}(A) = \text{relations}(A')$. Hence A' consistent with substitution $\sigma \Rightarrow A$ is consistent with substitution σ . So N preserves consistency as required.

Using adaptive matching

The following results allow us to work with adaptive matching (c.f. §4.3.3.5). Note that they do not require any conditions on our ability to evaluate relations. We show that if a conjecture is modified/proved using adaptive matching, then the modified conjecture can be proved with normal matching.

Lemma 8 (Adaptive matching \Rightarrow normal matching after update):

Given diagram A and reasoning program \underline{B} such that $A \simeq m \simeq \underline{B}$, $\underline{B} \in \text{leaves}(\underline{B})$ using Definition 4.3.3.6 (adaptive matching).

Let $\underline{B}' = \text{update}_a(\underline{B})$ and $B' = \text{update}_a(B)$. Then we have $A \simeq m \simeq B'$ using Definition 4.3.2.4 (normal matching).

This is clear from Definition 4.3.3.6, Definition 4.3.2.4 and Definition 4.3.3.7.

Lemma 9: Suppose diagram D is redrawn to give reasoning program \underline{D} using redraw rules Δ with adaptive matching. Let $D' = \text{root}(\underline{D})$. Then D' can be redrawn to give \underline{D} using Δ with normal matching.

This is obvious for simple/branch rules from Lemma 8 and Lemma 6. The only subtlety concerns animated rules. Suppose R is an animated rule used in drawing \underline{D} , then either R was applied before D' was created (and is therefore not needed to draw \underline{D} from D'), or Definition 4.3.3.7 guarantees that R can be applied as before.

Theorem 10: Adaptive matching with condition updating is sound: Suppose diagram D is redrawn using rules Δ to give reasoning program D with root diagram D' . If Δ are sound, then

$N:D' \mapsto \text{leaves}(D)$ is a sound rule.

This follows from Lemma 9 and Theorem 7.

Soundness of the meta rules

This follows quite simply from the results in the previous section, with the addition of the following lemmas regarding deleting relations and the elimination of 'dead' branches (those that end only in diagrams containing false):

Lemma 11: Deleting objects/relations is always sound

Let R be a deletion rule (c.f. §4.2.8) and D, D' be diagrams such that $D \xrightarrow{R} D'$.

Then $\mathbb{I}(D') \subset \mathbb{I}(D)$, so $\mathbb{I}(D)$ consistent with substitution $\sigma \Rightarrow \mathbb{I}(D')$ consistent with substitution σ .

Lemma 12: Suppose $N:D_0 \mapsto D_1 \dots D_n$ is a sound rule and $\text{false} \in \text{relations}(D_n)$. Then $N':D_0 \mapsto D_1 \dots D_{n-1}$ is sound.

Proof:

Suppose $D_0 \simeq n \simeq A_0$, $A_0 \xrightarrow{N} A_1 \dots A_n$ and A_0 is consistent with substitution σ . N

sound $\Rightarrow \exists i$ such that A_i is consistent with substitution σ .

$\text{false} \in \text{relations}(D_n) \Rightarrow \text{false} \in \text{relations}(A_n)$ hence A_n is not consistent.

Therefore $i < n$ and N' is sound.

Lemma 13 (merge is sound): Given a consistent reasoning program D , if D' is created by applying merge to D , then D' is consistent

Applying merge is equivalent to discarding dead branches (sound by Lemma 12), deleting those objects and relations not found in all other branches (sound by Lemma 11).

Technically, Lemma 13 only justifies using merge at the end of a proof, which is all we need. It does not justify using merge in the course of a proof, as this would invalidate the proof of Theorem 7 which was not proved for meta-rules. This restriction does not affect the range of the logic, but only the neatness of some proofs – and only more complicated proofs than we have produced. It is likely that Theorem 7 *does* extend to cover merge, which could then be used during proofs.

Lemma 14: extract is sound

This follows from Theorem 7 and Lemma 12.

Soundness of counter-example reasoning

A counter-example proof consists of a reasoning program \underline{D} constructed from a blank diagram D_0 using rules $\Delta \cup \{R\}$ such that $\forall D \in \text{leaves}(\underline{D})$, $\text{false} \in \text{relations}(D)$, from which we conclude that if Δ is sound then R is false. Counter-example reasoning can also use a special implicit inference rule (c.f. §4.3.5), which we now examine in Lemma 15.

Lemma 15: The rule $R = \text{“implicit}(r(x,y)) \text{ and specific}(x) \text{ and specific}(y) \text{ and accurate}(\mathcal{D}) \Rightarrow r(x,y)\text{”}$ preserves consistency

Suppose we have physical diagram \mathcal{D} with $r(x,y) \in \text{relations}(\mathcal{D})$ such that R can be applied to \mathcal{D} to give diagram $\mathcal{D}' = \{\text{objects}(\mathcal{D}), \text{relations}(\mathcal{D}) \cup \{r(x,y)\}\}$. We require that \mathcal{D} consistent $\Rightarrow \mathcal{D}'$ consistent.

Let X, Y be the objects in \mathcal{D} such that $\text{label}(X)=x$, $\text{label}(Y)=y$

\mathcal{D} accurate $\Rightarrow \exists D \in \mathbb{D}^{-1}(\mathcal{D})$ such that $\mathbb{M} \models \sigma_D(\mathbb{I}(D|\underline{D}))$ (where σ_D identifies objects in D with the identical objects in \mathbb{M})

Let x', y' be the domain objects in \mathbb{M} corresponding to x, y . Then $\mathbb{D}(x')=X$.

Similarly $\mathbb{D}(y')=Y$.

$\text{implicit}(r(x,y)) \Rightarrow r(x,y)$ is true for all x, y such that $\mathbb{D}(x)=X$, $\mathbb{D}(y)=Y$

$\Rightarrow r(x,y)$ is true for x', y'

$\Rightarrow \mathbb{M}$ is also a model for $\sigma_D(\mathbb{I}(D'|\underline{D}))$. So D' is consistent, and therefore \mathcal{D}' is also consistent.

Theorem 16: Counter-example proofs are sound

D_0 is blank and therefore trivially consistent. Sound rules preserve consistency by definition, hence if $\Delta \cup \{R\}$ is sound then \exists diagram $d_i \in \text{leaves}(\underline{D})$ such that D_i is consistent. But this is impossible, since $\text{false} \in \text{relations}(D_i)$. Hence Δ sound $\Rightarrow R$ unsound as required.

Soundness of conversion

Theorem 17 (Conversion Theorem): $R:D \rightarrow D'_1 \dots D'_k$ sound in Extended DDLA $\Leftrightarrow \mathbb{I}(R)$ sound in \mathbb{M}

Proof:

Suppose $R:D \mapsto D'_1 \dots D'_k$ is sound in Extended DDLA and consider

$$\mathbb{I}(R) = “\mathbb{I}(D) \Rightarrow Q(R). \mathbb{I}(D'_1) \vee \dots \vee \mathbb{I}(D'_k)”$$

This is sound if \forall substitutions σ such that $\sigma(\mathbb{I}(D))$ is true, $\exists i$ such that $\sigma(Q(R). \mathbb{I}(D'_i))$ is also true.

Suppose σ is such a substitution. Let $X_1 \dots X_m$ be the objects in $\sigma(\mathbb{I}(D))$. Let A be the diagram $(\{X_i\}, \text{relations}(D))$ (which is a diagram in EDDLA, though not necessarily in DDLA).

Note that $\sigma(\mathbb{I}(D))$ true $\Rightarrow A$ is consistent with substitution σ . Also $D \simeq n \simeq A$ using σ to define a matching.

Hence we can apply R to A to create diagrams $A'_1 \dots A'_k$ and $\mathbb{I}(A'_i) \Leftrightarrow \sigma(\mathbb{I}(D'_i))$

R sound $\Rightarrow \exists A'_i$ consistent with substitution $\sigma \Rightarrow \sigma(\mathbb{I}(D'_i))$ is true as required.

Conversely, suppose $\mathbb{I}(R)$ is sound.

Let $B, B'_1 \dots B'_k$ be diagrams such that $B \xrightarrow{R} B'_1 \dots B'_k$ with matching n , and B is consistent with substitution σ .

$\Rightarrow \sigma(\mathbb{I}(B))$ is true

$D \simeq n \simeq B \Rightarrow n(\mathbb{I}(D)) \Leftrightarrow \mathbb{I}(B)$, therefore $\sigma(n(\mathbb{I}(D)))$ is true

$\mathbb{I}(R)$ sound $\Rightarrow \sigma(n(\mathbb{I}(D'_1) \vee \dots \vee \mathbb{I}(D'_k)))$ is true

$\Rightarrow \sigma(\mathbb{I}(B'_1)) \vee \dots \vee \sigma(\mathbb{I}(B'_k))$ is true
 $\Rightarrow \exists i$ such that $\sigma(\mathbb{I}(B'_i))$ is true
 $\Rightarrow B_i$ is consistent with substitution σ as required.

5.2.3 Summary

This section has set out a semantics for DDLA, grounded in a standard model for Euclidean-space analysis. Working at the level of abstract diagram descriptions, we have shown that the inference mechanisms of DDLA are sound. This involved examining:

- 1) The basic inference mechanism of redraw rules
- 2) The use of meta rules
- 3) The use of short-cuts in counter-example reasoning (based on a procedure for evaluating the accuracy of a diagram). Note that part of this proof requires an in-depth examination of physical representations of the subset relation, which has been deferred to §5.3.

We have also shown that redraw rules can be converted into equivalent algebraic rules, and that if these conversions are sound then original redraw rule is sound. The converse is also true within a larger system we call Extended DDLA. This means that a diagrammatic proof of a diagrammatic theorem produced using redraw rules whose conversions are sound, constitutes a sound proof of the equivalent algebraic theorem.

5.3 The subset relation

Although there are many graphical aspects to DDLA, the subset relation is the main place where DDLA draws on the geometric properties of diagrams. It does so in three ways:

- 1) To implicitly represent statements of the form $A \subset B$
- 2) To automatically infer $A \subset C$ from $A \subset B, B \subset C$
- 3) When presenting counter-examples, a diagram with implicit subset relations guarantees the existence of a model with the same relations.

The use of implicit subsets – particularly its use in counter-example reasoning – is not as straightforward as it might seem. This section formalises how we define A inside B , and proves that this gives the desired properties – namely that “ A inside B ” can be reliably determined for our diagrams, and that being able to draw a diagram guarantees the existence of matching models.

Note: Some of the terminology used in this section overlaps with concepts in DDLA (e.g. *closure*). However this section is outside DDLA, and these terms do not refer to the DDLA definitions or usage.

5.3.1 Representing $A \subset B$

Recall from §5.1.1 that in order to use the diagram to represent subset relations, we need to be able to evaluate the implicit relation rule relating to subset. From §4.4.5, given an ideal diagram D , we have:

“ $\text{label}(A) \subset \text{label}(B)$ ” $\in \text{relations}(D)$ if A is clearly drawn inside B

In the light of §5.1.2 and assumptions (a) and (b), we can now give a more precise statement of this rule and show that it can be evaluated.

Notation

Power sets: If X is a set, write $P(X)$ for the power set of X

Flattening function: Given $A \subset P(X)$, define $U(A) = \bigcup_{a \in A} a$. U is a function $P(P(X)) \rightarrow P(X)$.

Paths: If x, y are points in $\mathbb{R}^2 \cup \{\infty\}$, let $p: x \rightarrow y$ denote that p is a path from x to y (i.e. p is a function, $p: [0, 1] \rightarrow \mathbb{R}^2$, $p(0) = x$, $p(1) = y$). A *simple* path is one that does not intersect itself, except possibly at its endpoints. Given a simple path p passing through points x, y , let $p: x \rightarrow y$ denote the section of path starting at x and ending at y .

Definitions

Our first three definitions are standard analysis concepts: closure under sequence limits, set border and set interior.

Definition 5.3.1.1: Set closure

Given an ideal set A , let $\text{closure}(A) = \{x \in \mathbb{R}^2 \mid \exists \text{ sequence } a_n \subset A . a_n \rightarrow x\}$

Definition 5.3.1.2: Set border

Given an ideal set $A \subset \mathbb{R}^2$, let $\text{border}(A)$ be the set of points making up its

border: $\text{border}(A) = \{a \mid a \in \text{closure}(A) \text{ and } \forall \epsilon > 0, B_\epsilon(a) \not\subset \text{closure}(A)\}$.

For physical sets $\mathbb{D}(A)$, let $\text{border}(\mathbb{D}(A)) = \mathbb{D}(\text{border}(A))$. Note that this differs from the definition for ideal sets, in that instead of a curve, it gives a 'blocky'

border composed of grid squares. For this to be well-defined we require:

$\forall A' \in \mathbb{D}^{-1}(\mathbb{D}(A)), \mathbb{D}(\text{border}(A')) = \mathbb{D}(\text{border}(A))$. This is trivial, since if $\mathbb{D}(A')$ has a different border from $\mathbb{D}(A)$ then A' cannot be in $\mathbb{D}^{-1}(A)$.

Definition 5.3.1.3: Set interior

Given a set A , let $\text{interior}(A)$ be the set of interior points:

$\text{interior}(A) = \{a \in A \mid a \notin \text{border}(A)\}$ for ideal sets.

$\text{interior}(A) = \{a \in U(A) \mid a \notin \text{border}(A)\}$ for physical sets.

Lemma 18: $\text{border}(\mathbb{D}(A))$, $\text{interior}(\mathbb{D}(A))$ can be calculated for all drawable sets A

Proof is by induction on the structure of A .

Let D be the diagram containing A .

Base case: Suppose A is a basic set

Then A is bounded so we can choose \mathbb{B} large enough that $\mathbb{D}(A)$ is contained inside \mathbb{B} and does not touch the borders of \mathbb{B} (and moreover since a finite collection of bounded objects is bounded, we can choose \mathbb{B} large enough such that all basic sets in D are drawn within \mathbb{B}).

A is represented by a simple closed curve a

(i.e. $\text{border}(\mathbb{D}(A)) = \mathbb{D}(A) = \mathbb{D}(a)$). By Assumption 1 (c.f. §5.1.1), $\mathbb{D}(A)$ is a set of squares in \mathbb{B} that the computer can calculate.

We can calculate $\text{interior}(\mathbb{D}(A))$ from $\mathbb{D}(A)$ by sorting $\mathbb{B} \setminus \mathbb{D}(A)$ into disjoint connected sets of squares. This method is inefficient but decidable, since \mathbb{B} is a finite set. Let E be the union of those sets which connect with the border of the diagram. E is the exterior of $\mathbb{D}(A)$ and $\text{interior}(\mathbb{D}(A)) = \mathbb{B} \setminus (E \cup \text{border}(\mathbb{D}(A)))$.

Step case 1: Suppose $A = f(B)$ or $A = f^{-1}(B)$

Then A is drawn as a basic blob set, and hence $\text{border}(\mathbb{D}(A))$ and $\text{interior}(\mathbb{D}(A))$ can be calculated as in the base case.

Step case 2: Suppose $A = A_1 \cap A_2$

Given the bitmap representations for A_1, A_2 , the bitmap representation for A can be calculated by the following formula:

$$\begin{aligned} \text{border}(A) = & (\text{border}(A_1) \cap \text{interior}(A_2)) \\ & \cup (\text{border}(A_2) \cap \text{interior}(A_1)) \\ & \cup (\text{border}(A_1) \cap \text{border}(A_2)) \end{aligned}$$

$$\text{and } \text{interior}(A) = \text{interior}(A_1) \cap \text{interior}(A_2)$$

Since the $\text{border}(A_i)$, $\text{interior}(A_i)$ are finite sets of squares, these formulae can be evaluated.

Step case 3: Suppose $A = A_1 \cup A_2$

$$\begin{aligned} \text{border}(A) = & (\text{border}(A_1) \setminus \text{interior}(A_2)) \cup (\text{border}(A_2) \setminus \text{interior}(A_1)) \\ \text{interior}(A) = & \text{interior}(A_1) \cup \text{interior}(A_2) \end{aligned}$$

Step case 4: Suppose A is a complement set, $A = B^c$

$$\begin{aligned} \text{Then } \text{border}(A) = & \text{border}(B) \\ \text{interior}(A) = & \mathbb{B} \setminus (\text{border}(A) \cup \text{interior}(B)) \end{aligned}$$

We formalise '*clearly inside*' as being inside with a clear border of empty space. We do so here in a bitmap-specific manner. See [58] for a formalisation that uses error margins instead.

Definition 5.3.1.4: Touching squares

Let a, b be two squares in \mathbb{B} . Say a, b are *touching squares* if $a \cap b \neq \emptyset$ (i.e. a touches itself and the 8 surrounding squares).

Given a square a and a set of squares B , say a *touches* B if $\exists b \in B$ such that a, b are touching squares.

Now given an ideal diagram D , we have:

Definition 5.3.1.5: 'Clear interior'

Given set A in ideal diagram D then the *clear interior* of A is recursively defined as follows:

If A is a basic set, then the clear interior of A is:

$$\text{interior}(\mathbb{D}(A)) \setminus \{b \in \mathbb{B} \mid b \text{ touches } \text{border}(\mathbb{D}(A))\}$$

If $A = A_1 \cap A_2$, then the clear interior of A is:

$$(\text{the clear interior of } A_1) \cap (\text{the clear interior of } A_2)$$

If $A = A_1 \cup A_2$, then the clear interior of A is:

(the clear interior of A_1) \cup (the clear interior of A_2)

If $A=B^c$, then the clear interior of A is:

$$(\text{border}(\mathbb{D}(B)) \cup \text{interior}(\mathbb{D}(B)))^c \setminus \{b \in \mathbb{B} \mid b \text{ touches } \text{border}(\mathbb{D}(A))\}$$

Definition 5.3.1.6: 'Clearly inside'

Given sets A, B in ideal diagram D , then A is clearly inside B iff $\mathbb{D}(A)$ is inside the clear interior of B .

Since interior and border can be calculated for all drawable sets, and consist of a finite set of squares in \mathbb{B} , the relation “ A clearly inside B ” can be evaluated from the bitmap representation. This satisfies the requirement in §5.1.1 that the computer can evaluate implicit relations rules.

Evaluating such rules using the bitmap also means that the computer is working from the same representation as the user. This helps justify assumption 3 in §5.1.1. If the user's judgement is not fine-grained enough to detect differences at the pixel level (e.g. if pixels are too small, A might be drawn inside B , but the user would not be able to perceive this), this evaluation can be adjusted to take that into account by demanding that A, B and $\min(\{|a-b| \mid a \in \mathbb{D}(A), b \in \text{interior}(\mathbb{D}(B))\})$ be of a minimum size. As specified in §4.3.2.1, cases where the relation is unclear can then be clarified with explicit statements. In practice, at the resolution we have been working at this has not been an issue.

5.3.2 Transitivity

Transitivity (i.e. that “ $A \subset B, B \subset C \Rightarrow A \subset C$ ”) is handled as an implicit inference rule (c.f. §4.4.6). For DDLA to be sound, we only require this rule to be sound. However, for this rule to make sense as a free ride, it should connect with some intuitive aspect of the visual representation. Here, transitivity makes sense because the inside relation used to represent it is also transitive. This is no coincidence: the representation for sets identifies a sets' members as its inside points (possibly plus border points). Hence a sets' subsets are precisely those that are drawn inside it (possibly plus sets drawn on or very near the border).

5.3.3 Diagram-model link

Recall that in Theorem 1, we claim that $\text{implicit}(A \subset B) \Rightarrow A' \subset B' \forall A' \in \mathbb{D}^{-1}(A), B' \in \mathbb{D}^{-1}(B)$ for a sub-class of DDLA diagrams (those which are believed accurate). This is needed to show that for that sub-class, the diagram guarantees the existence of an equivalent model. We now prove this result.

First let us consider how it might be false. For sets created by applying a function or an inverse function (i.e. $B=f(A)$ or $B=f^{-1}(A)$), this can happen quite easily as their representation is inaccurate (c.f. §4.4.5). For other sets, problems can arise because of roughness of drawing. Although normally roughness of drawing will only reduce $\text{interior}(B)$, it is possible in extreme cases for it to radically alter $\text{interior}(B)$ such that $\text{interior}(\mathbb{D}(B)) \not\subset \text{interior}(B)$. Figure 5.3 shows an example of this.

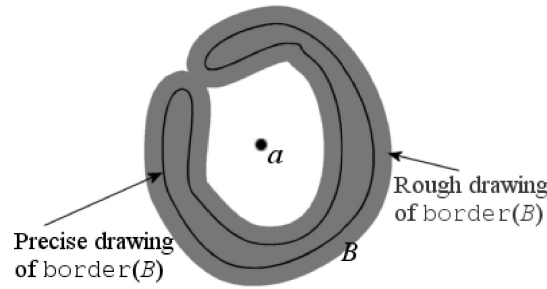


Figure 5.3. Example of $\text{implicit}(a \in B)$ but $a \notin B$

The condition that basic sets are star-shaped (c.f. §4.4.3) gives one way of preventing such cases from occurring. We show this via several lemmas. Lemma 19 proves that the 'closing eye' structure shown in Figure 5.3 is essentially the only way in which this issue can arise. Lemmas 20-24 then show that the restrictions we have applied in defining 'drawable sets' (c.f. §4.4.3) are sufficient to prevent such cases. Note that these results also generalise to representation systems where roughness of drawing is not caused by bitmaps (e.g. pen drawings, or representations where human eyesight is the key factor) – a result shown in [58].

Definition 5.3.3.1: Star-shaped

A set A is *star-shaped* (also referred to in some texts as *centred*) if \exists a point $c \in A$ such that $\forall a \in A$, the line $ac \subset A$.

Lemma 19: Given a connected set B with border b a simple closed curve, and a point $a \notin B$, $a \in \text{interior}(\mathbb{D}(b))$.

Then $\exists x, y \in B$, line $l = xy$ such that $\mathbb{D}(x), \mathbb{D}(y)$ are touching squares and either $a \in \text{interior}(b \cup l)$, or $\mathbb{D}(a)$ touches $\mathbb{D}(x)$ and $\mathbb{D}(y)$.

Proof:

Let M be a minimal subset of squares in $\mathbb{D}(b)$ such that

$a \in \text{interior}(U(\{B\} \cup M))$, and let $B_M = U(\{B\} \cup M)$

$a \in \text{interior}(B_M) \Rightarrow \exists$ closed curve $g \subset B_M$ such that $a \in \text{interior}(g)$.

M minimal $\Rightarrow g \cap U(m) \neq \emptyset \ \forall m \in M$

Claim 1: $U(M)$ is connected.

Suppose false: $\Rightarrow \exists M_1, M_2 \subset M$ such that $U(M_1), U(M_2)$ are not

connected. Let $M_3 = U(M_1) \cup U(M_2)$. Let $B' = B_M \setminus (M_3 \setminus B)$ and note that

B connected, M a set of squares connected to $B \Rightarrow B'$ connected.

$U(M_i)$ not connected, g a loop $\Rightarrow \exists$ points $x, y \in g \setminus M_3$ such that

$g: x \rightarrow y$ passes through M_1 , $g: y \rightarrow x$ passes through M_2

Let $g_1 = g: x \rightarrow y$, $g_2 = g: y \rightarrow x$

B' is connected, and $x, y \in B'$, therefore \exists curve $g' \subset B'$, $g': x \rightarrow y$

Now $g_i \cup g'$ are closed curves, $g_1 \cup g' \cap U(M_2) = \emptyset$ and $g_1 \cup g' \cap U(M_2) = \emptyset$

Also $\text{interior}(g) \subset (\cup_i \text{interior}(g_i \cup g') \cup g')$

$g' \subset B' \Rightarrow a \notin g'$, hence $a \in \text{interior}(g_1 \cup g')$ or $a \in \text{interior}(g_2 \cup g')$

WLOG say $a \in \text{interior}(g_1 \cup g')$

But $a \in \text{interior}(g_1 \cup g')$, M minimal $\Rightarrow (g_1 \cup g') \cap m \neq \emptyset \ \forall m \in M$. So this contradicts the minimality of M , hence M must be connected as claimed.

Figure 5.4 gives an example illustrating this situation.

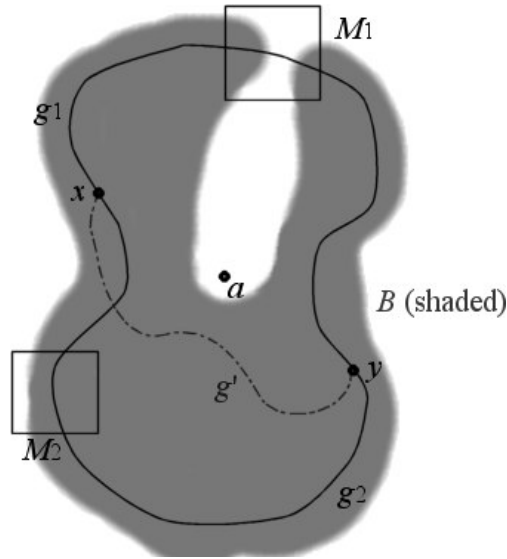


Figure 5.4. Disconnected M . The contradiction in this example is that $M_2 \subset M$ is not necessary for $a \in \text{interior}(B_M)$, and hence we get $g_2 \notin B$.

Claim 2: $|M| < 3$

Suppose false, and let m_1, m_2, m_3 be distinct squares in M

Let x_i be points in $B \cap g \cap m_i$

Let g_1 be the portion of g joining x_1 to x_2 , and g_2 the portion of g joining x_2 to x_3

g a simple curve $\Rightarrow g_1, g_2$ are disjoint except for x_2

Now B connected $\Rightarrow \exists$ curves $g_1', g_2' \subset B$ such that g_1' joins x_1 to x_2 , g_2' joins x_2 to x_3 .

$g_1 \cup g_1'$ is a closed curve. Moreover, we have $a \in \text{interior}(g_1 \cup g_1')$, since otherwise we would have $a \in \text{interior}(g_1' \cup g_2 \cup g_3)$, which would contradict the minimality of M .

Similarly $a \in \text{interior}(g_2 \cup g_2')$

So $a \in \text{interior}(g_1 \cup g_1') \cap \text{interior}(g_2 \cup g_2')$

This is illustrated in Figure 5.5.

Now suppose g_1', g_2' intersect at a point y

Then let f be the path $f: x_2 \rightarrow y \rightarrow x_2$ formed from g_1', g_2'

But then $a \in \text{interior}(f)$, which implies $a \in \text{interior}(B)$.

Contradiction, hence $|M| < 3$

Therefore $g_1' \cap g_2' = \{x_2\}$.

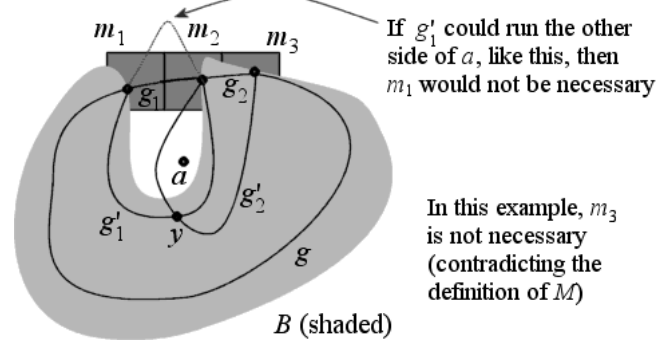


Figure 5.5. $|M| > 2$

But then consider the path $f': x_2 \rightarrow x_3 \rightarrow x_1 \rightarrow x_2$ made from g_2' , $g \setminus g_i$, g_1' respectively. This has $a \in \text{interior}(f')$ and $f' \subset B_M \setminus m_i$ – contradicting the minimality of M . Hence $|M| < 3$.

Note that M connected, $|M| < 3 \Rightarrow M$ is composed of touching squares

Now let x, y be the entry and exit points of g in M .

M composed of touching squares, $|M| < 3 \Rightarrow M$ can be in only two arrangements: a 1x2 rectangle, or diagonally touching squares.

Suppose M is a 1x2 rectangle. This is convex, therefore the line

$$xy \subset M$$

$$\text{Hence } \text{interior}(g) \setminus \text{interior}((g \setminus M) \cup xy) \subset M.$$

$$a \notin M \Rightarrow a \notin \text{interior}(g) \setminus \text{interior}((g \setminus M) \cup xy)$$

Hence $(g \setminus M) \cup xy$ is a closed curve in B_M such that

$$a \in \text{interior}((g \setminus M) \cup xy)$$

$$g \setminus M \subset B, \text{ so } a \in \text{interior}(B \cup xy) \text{ as required}$$

Otherwise M is composed of diagonally touching squares

Consider the 2x2 rectangle M' such that $M \subset M'$.

If $a \in M'$ then a is in a touching square to M and we are done.

Otherwise, M' convex, $a \notin M' \Rightarrow a \in B \cup xy$ as for the 1x2 case.

□

Lemma 20: Given a triangle Δxyz such that x, y are in touching squares in \mathbb{B} , then there are no points that are clearly inside Δxyz .

Proof:

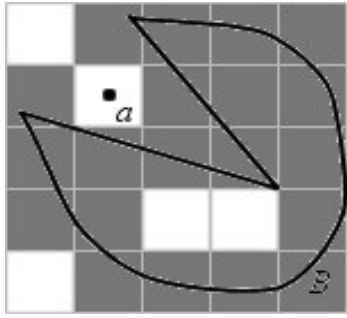
Suppose such a point exists. Then there is a 3×3 block of squares lying within Δxyz (the square containing the point plus the touching squares).

So $|x-y|, |z-x|, |y-z| > 3$.

But x, y in touching squares $\Rightarrow |x-y| \leq \sqrt{8} < 3$, which is a contradiction, hence no such points exist.

Note that this result would fail if the definition of 'clearly inside' did not require a border of empty space. Figure 5.6 gives a counter-example where $a \notin B$ but

$$\mathbb{D}(a) \subset \text{interior}(\mathbb{D}(B))$$



The black curve is the border of B ,
the shaded squares are $\mathbb{D}(B)$.

Figure 5.6. Example of $a \notin B$ but $\mathbb{D}(a) \subset \text{interior}(\mathbb{D}(B))$.

Lemma 21: Given drawable basic set B with clear interior B' , then $B' \subset B$

Proof:

Suppose false $\Rightarrow \exists a \in B', a \notin B$

$\Rightarrow a \in \text{interior}(\mathbb{D}(B))$ and $\mathbb{D}(a)$ does not touch any squares in $\mathbb{D}(B)$

So by Lemma 19, \exists points $x, y \in B$ such that $a' \in \text{interior}(B \cup xy)$

All of DDLA's basic sets are star-shaped, hence \exists point $c \in B$ such that lines $xc, yc \subset B$

Let $g \subset B \cup xy$ be a simple closed curve such that $a \in \text{interior}(g)$

Suppose $a \notin \Delta xyc$. But then $a \in \text{interior}((g \setminus xy) \cup xc \cup yc) \subset B$. This contradicts $a \notin B$, hence $a \in \Delta xyc$

But by Lemma 20 there are no such points. This is a contradiction, hence we are done.

Lemma 22: \forall drawable sets A , $\text{interior}(\mathbb{D}(A^c)) \subset A^c$

Note that $\text{border}(A) \subset \text{border}(\mathbb{D}(A))$.

$x \notin A^c \Rightarrow x \in A \Rightarrow x \in \text{border}(A)$, or \exists closed curve $f \subset \text{border}(A)$ such that $x \in \text{interior}(f)$

But then either $x \in \text{border}(\mathbb{D}(A))$ or $f \subset \text{border}(\mathbb{D}(A)) \Rightarrow x \in \text{border}(\mathbb{D}(A)) \cup \text{interior}(\mathbb{D}(A))$
 $\Rightarrow x \notin \mathbb{D}(A)^c$

Lemma 23: Given drawable set B such that B was not constructed using function or function inverse application, let B' be the clear interior of B . Then $B' \subset B$

Proof by induction on the structure of B

Suppose B is a basic set

True by Lemma 21.

Suppose $B = B_1 \cap B_2$

Then the clear interior of $B = (\text{clear interior of } B_1) \cap (\text{clear interior of } B_2) \subset B_1 \cap B_2$ by induction

Suppose $B = B_1 \cup B_2$

Then the clear interior of $B = (\text{clear interior of } B_1) \cup (\text{clear interior of } B_2) \subset B_1 \cup B_2$ by induction.

Suppose $B = A^c$

Then the (clear interior of B) $\subset \text{interior}(\mathbb{D}(A)^c) \subset A^c$

Theorem 24: Given ideal diagram D such that there are no function objects in D , and sets A, B in D , then $\text{implicit}(A \subset B) \Rightarrow A' \subset B'$

$\forall A' \in \mathbb{D}^{-1}(\mathbb{D}(A)), B' \in \mathbb{D}^{-1}(\mathbb{D}(B))$

Suppose we have sets A, B, A', B' such that $\text{implicit}(A \subset B)$ and $A' \in \mathbb{D}^{-1}(\mathbb{D}(A)), B' \in \mathbb{D}^{-1}(\mathbb{D}(B))$

Now B, B' have the same clear interior, and $\mathbb{D}(A)=\mathbb{D}(A')$ have the same border

$\text{implicit}(A \subset B) \Rightarrow A$ is clearly inside B

$\Rightarrow \mathbb{D}(A) \subset (\text{clear interior of } B)$

$\Rightarrow \mathbb{D}(A') \subset (\text{clear interior of } B')$

$\Rightarrow \mathbb{D}(A') \subset B'$ (by Lemma 23)

$\Rightarrow A' \subset B'$ as required.

5.4 Examples of showing specific rules are sound

The full set of soundness proofs for DDLA are given along with the axioms in Appendix 12. Here we give three example soundness proofs; one for each type of redraw rule. Since we want DDLA proofs to carry over to the algebraic version of analysis, we show soundness by conversion for simple/branch rules, and soundness in EDDLA for animated rules.

5.4.1 Simple rule: open set definition

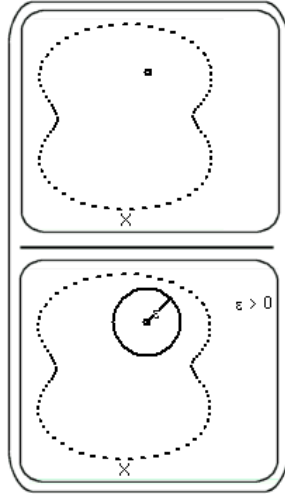


Figure 5.7. Open set definition.

Let $R:D_0 \mapsto D_1$ be the simple redraw rule shown in Figure 5.7.

The implicit relations in D_0 are $\{\text{set}(X), \text{point}(x), x \in X\}$, and the explicit relations are $\{\text{open}(X)\}$.

The implicit relations in D_1 are $\{\text{set}(X), \text{point}(x), x \in X, \text{ball}(B), \text{line}(\epsilon), B \subset X,$

$\text{centre}(B,x), \text{radius}(B,\epsilon), \epsilon > 0\}$. The explicit relations are $\{\text{open}(X)\}$.

Hence $\mathbb{I}(R) = “\mathbb{I}(D_0) \Rightarrow Q(R) \text{ labels}(D_1 \setminus D_0). \mathbb{I}(D_1)”$

$$= “\{\text{set}(X), \text{point}(x), x \in X, \text{open}(X)\} \Rightarrow \exists B, \epsilon \{\text{set}(X), \text{point}(x), x \in X, \text{ball}(B), \text{line}(\epsilon), B \subset X, \text{centre}(B,x), \text{radius}(B,\epsilon), \epsilon > 0\}”$$

which simplifies to:

$$\mathbb{I}(R) = “\text{set}(X), \text{point}(x), x \in X, \text{open}(X) \Rightarrow \exists B, \epsilon. \epsilon > 0, B_\epsilon(x) \subset X”$$

But this is just the standard definition for an open set “ $\text{open}(X), x \in X \Rightarrow \exists \epsilon > 0. B_\epsilon(x) \subset X$ ” (noting that $\forall x, \epsilon \exists B = B_\epsilon(x)$). This definition is part of \mathbb{A} , and therefore assumed to be sound (c.f. §5.1.1). Hence R is sound by Theorem 17.

5.4.2 Branch rule: set union definition

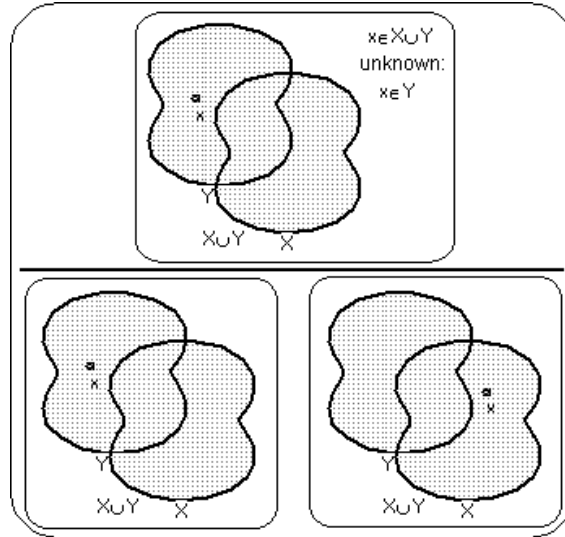


Figure 5.8. Set union definition.

Let $R: D_0 \mapsto D_1, D_2$ be the branch redraw rule shown in Figure 5.8.

The implicit relations in D_0 are $\{x \in Y, x \in X \cup Y\}$. The explicit relations are $\{\text{unknown}(x \in Y)\}$.

Hence $\text{relations}(D_0) = \{x \in X \cup Y\}$. In the consequent diagrams D_1, D_2 we have $\text{relations}(D_1) = \{x \in X \cup Y, x \in X\}$, $\text{relations}(D_2) = \{x \in X \cup Y, x \in Y\}$ and $\text{labels}(D_i \setminus D_0) = \emptyset$.

Hence $\mathbb{I}(R) = “\mathbb{I}(D_0) \Rightarrow Q(R) \cup_i \text{labels}(D_i) \setminus \text{labels}(D_0) . \mathbb{I}(D_1) \vee \dots \vee \mathbb{I}(D_n)”$

$$= “\{x \in X \cup Y\} \Rightarrow \exists . (x \in X \cup Y, x \in X) \vee (x \in X \cup Y, x \in Y)”$$

where the \exists quantifier is not quantifying over any variables, since

$$\cup_i \text{labels}(D_i) \setminus \text{labels}(D_0) = \emptyset.$$

Which simplifies to:

$$\mathbb{I}(R) = "x \in X \cup Y \Rightarrow x \in X \vee x \in Y"$$

Again, this is just the standard definition for set union (which is part of \mathbb{A} and therefore sound by assumption). Hence R is sound by Theorem 17.

5.4.3 Animated rule: recognising $Y \subset X$

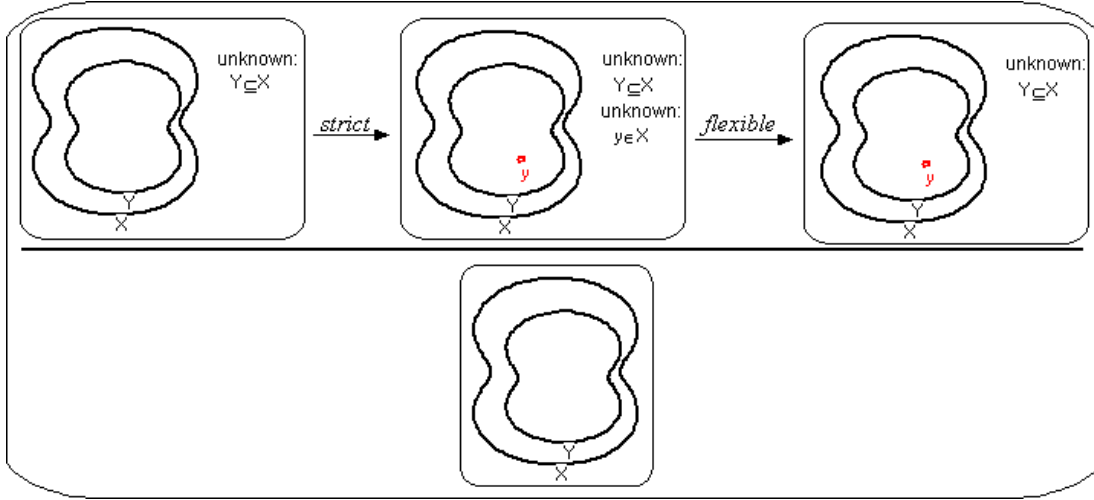


Figure 5.9. Subset definition.

Let $R: T_1-s-T_2-f-T_3 \mapsto T'$ be the rule shown in Figure 5.9 for recognising $Y \subset X$. We will show that it is sound by considering the possibility of an unsound application (i.e. an application that created an inconsistent diagram from a consistent one).⁵⁷

Proof that R is sound

Suppose R is not sound.

$\Rightarrow \exists \underline{D}, \underline{D}'$ such that D_0 is consistent, \underline{D} was drawn from D_0 using sound rules Δ , $\underline{D} \xrightarrow{R} \underline{D}'$, but \underline{D}' is inconsistent.

Let $\underline{T} \simeq m \simeq \underline{D}$. Without loss of generality, let the object labels in $m(\underline{T}) \subset \underline{D}$ be the same as those in \underline{T} and assume y' is not used as an object label in \underline{D} .

Let $D = \text{target}(R, \underline{D})$ and $D' = R(D)$.

\underline{D} consistent, \underline{D}' inconsistent $\Rightarrow D$ consistent, D' inconsistent (since all other branches of \underline{D} , if any, are unchanged by the application of R and must therefore have been inconsistent already).

⁵⁷ The process we use here and in appendix A for checking the soundness of animated rules is slightly laborious. It could probably be replaced by a general lemma covering the interpretation of animated rules.

$\underline{D} \xrightarrow{R} \underline{D}' \Rightarrow \text{changes}(D, D') = \{\emptyset, \{y\}, \{Y \subset X\}, \emptyset\}$ (that is, the universally quantified point y is deleted, and the relation $Y \subset X$ is added to diagram D')

\underline{D} drawn using sound rules $\Rightarrow D$ is consistent. Hence the statement $Y \subset X$ must be inconsistent with D .

D consistent, hence there exist models for D (note: if D is also accurate, then $\text{objects}(D)$ is a model for D). Let $M(D)$ be a model for D with the same object

labels. $Y \subset X$ inconsistent with $D \Rightarrow \exists y' \in M(D)$ such that $y' \in Y$ and $\text{not}(y' \in X)$.

Let \underline{A} be the reasoning program $\underline{D}[y'/y]$. But then by Lemma 6, \underline{A} can be drawn using Δ in the same way as for \underline{D} . Hence $\text{target}(R, \underline{A})$ has $y' \in X$.

Δ sound $\Rightarrow \underline{A}$ is consistent, $\therefore y' \in X$. This is a contradiction, hence R is sound.

Note that this rule could not be implemented using a simple redraw rule. In proving it is sound, we need to be able to perform a substitution (y' for y) and know that the method used to show $y \in X$ will still work for y' . This is possible here because the animated pre-condition controls the creation of x and thus prevents us from 'cheating' in demonstrating $y \in X$.

5.5 Summary

This chapter continues the formalisation work started in §4.3, defining several concepts relating to soundness. We have specified a semantics for our logic. This allowed us to show that the inference mechanisms of DDL are sound, and to give tools for showing that the individual redraw rules of DDLA are sound. The semantics explicitly links DDLA to its domain, and – via the concept of accurate diagrams – provides a way of using that link in DDLA reasoning.

This chapter has also examined the assumptions underlying the project, concluding that we cannot do away with the requirement for diagrams to be 'clear', where clear means that objects and relations can be reliably identified. This could potentially be formalised by developing an idea of valid and invalid diagrams based on clarity of representation judged from bitmap comparisons. However, having examined the clarity requirement in the context of DDLA representations, we conclude that it is reasonable.

We have paid particular attention to the subset relation, analysing its representation at the level of the physical drawing. This is because this relation is used as a reasoning short-cut as well as a convenient representation. Our treatment of this relation reveals the potential for misleading diagrams in extreme cases, and has demonstrated that these cannot occur in DDLA.

6 Evaluation

We begin by restating the aims of this project, and considering the ways in which such a project can be said to succeed or fail. This then suggests the following hypothesis for evaluation: “Computerised diagrammatic proofs are possible for analysis problems, and may be easier (in some sense) than algebraic proofs”. This hypothesis requires independent evaluations on the criteria of (1) soundness, (2) coverage of the domain and (3) ease of use. The first criteria is evaluated by analytic studies, the second is a reasoned judgement, whilst the third criterion is evaluated by an empirical study. The results loosely support the hypothesis.

6.1 Evaluation criteria

6.1.1 Project aims

The overall aim of this project is to investigate the potential for applying a diagrammatic approach to mechanised reasoning. This is motivated by the wider aim of producing theorem provers which are easier for people to understand. The assumption underlying this project is that, for some domains, diagrammatic reasoning is easier to understand for at least a significant number of people.

Since there has been little research to date in this area, and none in the domain we focused on, the first stage of the project was exploratory. Its aim was to develop sufficiently powerful diagrammatic techniques to tackle analysis problems. The techniques developed should have the potential for practical application in mathematics teaching, where, we hope, they will complement conventional methods (although the development of such an application was beyond the scope of this project).

These aims led to the adoption of two goals:

- 1) To develop a formal logic that uses diagrammatic reasoning to solve problems in real analysis.

- 2) To make a case for this logic having advantages over the conventional algebraic approach.

6.1.2 Ways of evaluating

We now consider how a project with these aims should be evaluated. There are levels of success and failure. The basic criterion for such a project is the development of a sound diagram logic. Given this, we identify several different axes along which the logic produced should be measured:

- 1) The range of the logic.

The logic should cover an interesting range of theorems. This is important to both of the project's goals. Clearly, there are degrees of success here. It is important that range is not achieved at the cost of making the logic unwieldy.⁵⁸

- 2) Implementation.

Computer implementations provide a safeguard against hidden assumptions that might make the logic unsound. A reasoning method can appear analytic, whilst actually requiring intuitive leaps, or 'magic steps' to work. By demonstrating that DDLA can be carried out by a computer we show that it does not require any such magic steps. The key thing is not the quality of the implementation (which is largely a factor of the time invested), so much as the fact that the logic *can* be implemented. It is possible that a logic could be developed that looks good on paper, but some of the steps prove impossible, intractable, or just too difficult/time-consuming to break down into proper algorithms. This would be a serious failure, although not necessarily a complete one, as the work done might still have developed the theory of diagrammatic reasoning in useful directions.

- 3) Usability.

The motivation for developing diagrammatic reasoning is mainly that it should be easier for people to understand than algebraic equivalents, so usability is key to all work in diagrammatic reasoning. Note that there are scenarios where - although in general -

⁵⁸ The 'normal' test for this would be to compare the number of rules required to the range achieved. Too many rules would suggest that the logic lacks flexibility and does not scale well. It is not clear that this measure is the most appropriate for logics intended for human use. An empirical measure (i.e. based on experimental study of use) would be better, although considerably more time consuming.

people find the diagrammatic proofs are no easier to understand, the project could still be considered successful. These are:

- a) A pattern can be found whereby certain types of student perform better with diagrammatic proofs, whilst others perform better with algebra. This was the result of Stenning *et al*'s research with HYPERPROOF [51].
- b) A pattern can be found of which proofs are harder/easier to understand with diagrams. In either of these cases, the project would have demonstrated that diagrammatic reasoning is useful in this domain, albeit only in certain circumstances. Moreover, establishing conditions when diagrams are and are not useful would be a worthwhile result in itself.

6.1.3 Hypothesis for evaluation

Considering the ways in which the project should be measured, suggests the following hypothesis for evaluation:

Show that diagrammatic reasoning in the domain of real analysis is possible such that:

- 1) It is sound (subject to reasonable assumptions).*
- 2) A reasonable range is achieved without requiring an unwieldy set of rules.*
- 3) It can be implemented on a computer.*
- 4) Some of the proofs produced are 'easier', either to produce or understand, for a significant proportion of people.*

The first three criteria relate to the goal of developing a formal logic that uses diagrammatic reasoning to solve problems in real analysis. The section 'Mathematical Evaluation' tackles how we measure them. The fourth criteria relates to the goal of producing a pedagogical tool. How we measure this is tackled in the section 'Empirical Evaluation'. The criteria are largely independent, and it is not necessary for the project to succeed at all of them. We consider criteria (2) and (3) to be desirable, whilst (1) and (4) are crucial.

We summarise this hypothesis as “Computerised diagrammatic proofs are possible for analysis problems, and may be easier (in some sense) than algebraic proofs”.

6.2 Mathematical evaluation

6.2.1 Soundness

There are two aspects to soundness: soundness of the reasoning framework, and soundness of the individual reasoning rules. Soundness of inference in DDL was shown in chapter 5. The soundness of the DDLA rule-set is shown in Appendix A. Both of these are demonstrated to the level of textbook mathematical proof, subject to certain assumptions which we restate here:

- 1) That the standard approach to Euclidean-space analysis is sound.
- 2) That the computer can produce reliable external representations.

Which we broke down into smaller assumptions:

- a) The surface on which diagrams are drawn obeys Euclidean plane geometry upto detectable differences.
 - b) We can divide this surface into a grid such that the computer can draw domain objects on it upto the accuracy of the grid.
 - c) The computer can represent all explicit relations.
 - d) The computer can evaluate which relations will be implicit relations in the external representation (that is, it can correctly apply the implicit relation rules).
- 3) That the user can correctly identify graphic objects and their labels.
 - 4) That the user can correctly assess observed relations.

These assumptions were discussed in §5.1.1, where we concluded that they are reasonable. That discussion revealed that assumptions 3 and 4 require a further assumption (or restriction) that diagrams be of sufficient clarity for the user to parse. However this seems a reasonable assumption (and is an implicit assumption in any non-discrete diagrammatic reasoning systems). We therefore conclude that DDLA is a sound logic for theorem proving in the domain of mathematical analysis.

6.2.2 Range

The range of DDLA is a small subset of that which can be represented/proved algebraically. This was inevitable given the size of the subject. We explore whether DDLA covers an interesting subset. For those areas not covered, we try to judge whether DDLA can

potentially be extended to cover them, or whether the lack of coverage is due to a structural limitation of our approach.

Range of representation

Logical statements

The inclusion of algebraic statements within DDL diagrams allows for a vast potential range of expression. DDL reasoning programs can express arbitrary conjunctions, disjunctions and negations of predicates with any (single) nesting of quantifiers. DDL rules can express any statement of the form $P \Rightarrow Q$ where P is a conjunction of predicates with any (single) nesting of quantifiers, and Q is a disjunction of conjunctions of predicates with one quantifier.

However, given that DDL is intended for human use, the theoretical range is possibly not as important as the range for which DDL is of practical value. This is hard to measure, since it is not so much a property of the logic, as of human understanding. Nor is it static; the 'useful range' will to some extent depend on the task being performed, and the abilities and training of the user. This 'practical range' is perhaps best shown by demonstration (i.e. by building logics in DDL, such as DDLA, and showing that they can be used).

Domain objects

DDLA provides specialised representations for several different types of domain object and can represent a vast range of different objects. However it has important limitations. There are important classes of sets that cannot be represented, including sets of measure 0 (i.e. sets composed of isolated points) and sets with an infinite number of discontinuities (e.g. \mathbb{Q}). There are also interesting functions that cannot be represented, including limit functions and non-integrable functions. The most serious limitations of DDLA though, are:

- 1) That it does not contain the concept of the natural numbers.
- 2) That it cannot define specific sequences (though it can deal with generic ones).

Analysis concepts

DDLA defines a range of analysis concepts, including: open and closed sets, continuous functions and convergent sequences. It does not cover the important concepts of differentiation, integration and uniform convergence. The concept of a uniformly continuous

function can be defined in DDLA, but we did not find any interesting results regarding it that can be proved within DDLA.

Range of proof

DDLA is not a complete logic. Amongst other limitations, it lacks the ability to handle certain types of proof. These include proofs involving recursion (e.g. proofs using repeated bisection). Unfortunately, we have not found a characterisation for the class of theorems it does cover. This forces us to evaluate range of proof in a 'soft' subjective manner. First we look at DDLA's performance on some sample exam papers. We then consider the list of theorems for which we have found DDLA proofs.

An empirical test of range

Quantifying the range of an incomplete reasoning system is not easy. As a crude metric, we examined how well DDLA could perform in Analysis exams. For this, we used the past exam papers from 'Fundamentals of Analysis', a 2nd year undergraduate course at Edinburgh University [65]. Three papers were available: the 2001 exam, the 2002 exam and a specimen exam for 2003.

The results of this exercise were disappointing: DDLA scored 17% (2001), 17% (2002) and 25% (2003). By far the largest obstacle was DDLA's lack of a mechanism for reasoning about specific sequences. Note that DDLA was not designed with this particular course in mind, and has not been adapted to fit this course in any way.

Theorems proved

We now look at the theorems for which we have produced proofs in DDLA. This is – presumably – only a representative list of what can be proved in DDLA and not an exhaustive one.

General theorems

- $(A \text{ open} \Rightarrow f^1(A) \text{ open}) \Rightarrow f \text{ continuous}$
- $f \text{ continuous}, A \text{ open} \Rightarrow f^1(A) \text{ open}$
- $X, Y \text{ open} \Rightarrow X \cap Y \text{ open}$
- $X, Y \text{ open} \Rightarrow X \cup Y \text{ open}$
- $X \text{ closed}, x_n \rightarrow x \text{ a convergent sequence}, x_n \in X \Rightarrow x \in X$

- f continuous, $x_n \rightarrow x$ convergent $\Rightarrow f(x_n) \rightarrow f(x)$
- X, Y closed $\Rightarrow X \cap Y$ closed
- X, Y closed $\Rightarrow X \cup Y$ closed
- f, g continuous $\Rightarrow fog$ continuous
- f, g continuous $\Rightarrow f+g$ continuous
- f, g continuous $\Rightarrow f.g$ continuous
- f decreasing and surjective $\Rightarrow f$ continuous

Specific theorems

- $B_r(x)$ open for $r > 0$
- $f(x) = 1/x$ continuous
- any given polynomial (defined over \mathbb{R}^+) is continuous (provable by recursion on function structure – DDLA lacks a discrete generalisation mechanism, such as proof-by-induction, to give the general-case proof).

'Small' theorems

In proving the theorems listed above, we have also proved many small lemmas. Some examples are:

- $r < r' \Rightarrow B_r(x) \subset B_{r'}(x)$
- $A \subset X, A \subset Y \Rightarrow A \subset X \cup Y$
- f decreasing, $f(x) > f(y) \Rightarrow x < y$
- $f(x) = x$ continuous

Counter-example proofs

- $\text{not}(\forall A, X, Y. A \subset X, A \subset Y \Rightarrow A \subset X \cap Y)$

Summary

This list contains a spread of diverse and non-trivial conjectures. It therefore demonstrates that DDLA does cover a moderately interesting range. However the 'big theorems' of analysis, such as the intermediate value theorem or results regarding differentiation and integration are noticeable by their absence.

The paucity of counter-example proofs means that the suitability of DDLA for reasoning about counter-examples has *not* yet been demonstrated (especially given that having diagrammatic support for such reasoning complicates the formalisation of DDLA). This aspect of DDLA must therefore be regarded only as an interesting line for future work.

6.3 Computer implementation: Dr.Doodle

This section describes the implementation of DDLA as a program called **Dr.Doodle**, which is included on the CD-ROM accompanying this thesis. The description is given at the user-level, focusing on design issues rather than algorithmic/programming ones.

We first cover the motivation behind producing this program, showing that it is a necessary part of this project. We then discuss how faithful **Dr.Doodle** is to DDLA. **Dr.Doodle** is not perfectly faithful to the specification for DDLA; there are elements it does not implement, and areas where it extends DDLA. However it is faithful enough to provide a good platform for evaluating DDLA. §6.3.3 gives an overview of the user interface design (illustrated with screenshots) and the underlying theorem prover.

In order to compare DDLA with a conventional logic, **Dr.Doodle** is built to operate with diagrams or algebra (c.f. §6.3.4). The algebraic mode is designed to be a 'fair' competitor to DDLA. I have therefore also put considerable effort into the design of the algebraic mode.

6.3.1 Motivation

The development of **Dr.Doodle** was motivated by several considerations.

Firstly, as noted in §6.1.2, a computer implementation provides a safeguard against developing a logic that looks good on paper, but in fact contains hidden complexities, requiring intuitive leaps to make it work.

Implementation is also crucial to the second, pedagogical, project goal. An interactive theorem prover allows students to both learn and be tested using DDLA. This provides a platform that can be used to run experiments, allowing us to evaluate the claim that diagrammatic reasoning is (in some sense) better than conventional logic for this domain. These experiments would not be feasible without a computer implementation of DDLA. The

experimental subjects would probably not be familiar with formal logic,⁵⁹ and explaining how a rewrite rule system works would be hard enough in itself. It is easier and better to present them with a computer program where the rigid nature of the reasoning is both expected and enforced. Also, a proof in DDLA is not given by the finished diagram, but by the drawing process. Capturing this objectively in a pen-and-paper experiment would be difficult.

A computer implementation also provides a good way of demonstrating our ideas. Moreover, the project aims were motivated by the idea of producing teaching software. **Dr.Doodle** provides a base upon which such software can be built.

6.3.2 DDLA and **Dr.Doodle**

Limitations

The main limitation of **Dr.Doodle** with respect to the DDLA specification is that it does not implement adaptive matching and condition updating (definition 4.3.3.6). Adaptive matching is an interesting aspect of DDLA, potentially useful for exploring conjectures. However our evaluation experiments were limited to short lessons with highly structured problems (c.f. §6.4). Adaptive matching was therefore not necessary.

The other limitation is a minor restriction of the objects which can be created within **Dr.Doodle**. **Dr.Doodle** cannot create sets from the cross product of intervals, and whilst **Dr.Doodle** can create basic rectangular area blocks, it cannot create compound area blocks as specified in §4.4.3. These limitations slightly restricts the reasoning that can be performed using **Dr.Doodle**, but do not affect its behaviour within these restrictions.

Both these limitations are the result of focusing the work on **Dr.Doodle** on more important aspects, rather than any theoretical difficulties.

Extensions

Dr.Doodle extends DDLA in three ways.

⁵⁹ Formal logic is not included in the Edinburgh University undergraduate mathematics syllabus, except via optional outside courses [66].

- 1) It allows object labels to have meaning (i.e. object labels obey a compositional semantics). 'Non-basic' objects such as $X \cap Y$ are (typically) labelled by the program according to how they are created. Under DDLA, they should be labelled with an atomic token (e.g. Z), with the relation $Z = X \cap Y$ being explicitly stated. In **Dr.Doodle**, such a set would be labelled $X \cap Y$, and no explicit relations would be added. This extension is implemented at a cosmetic level (the program chooses ' $X \cap Y$ ' as the atomic token labelling the set, and censors the statement ' $X \cap Y = X \cap Y$ ').
- 2) It can hide object labels. This is done in some of the rules to make the diagram clearer by reducing clutter.
- 3) **Dr.Doodle** highlights inaccurate relations by drawing them in mauve. This is done to draw attention to the inaccuracy. Inaccurate relations are generally avoidable, except when proving an inconsistency. However they often arise when a case split is performed, because typically the example initially drawn does not represent all cases.

It should be clear that these extensions improve diagram appearance without affecting the logic in any serious way.

6.3.3 System design

Dr.Doodle consists of a graphical user interface linked to an underlying theorem prover. There is considerable interaction between these two parts, which is described below.

The Theorem Prover (TP)

The theorem prover aspect of **Dr.Doodle** performs several different tasks:

- 1) Finding valid matches between diagrams
- 2) Applying redraw rules (note that this often requires input from the user to specify the appearance of new objects)
- 3) Detecting implicit relations
- 4) Performing implicit inferences
- 5) Detecting and 'creating' emergent objects
- 6) Checking proofs

We now give a brief description of the modules that perform these tasks.

Diagram Matching

This module finds matches between diagrams, as specified by definition 4.3.2.4. This is a constraint satisfaction problem, and is solved by a search. In the worst case the search is $O(m^n)$ where n is the number of graphic objects in the source diagram and m is the number of graphic objects in the target diagram. The use of heuristics – tweaked during program testing – leads to reasonable performance for the problems used in the empirical experiments and for small-medium sized diagrams.

Often there are several valid matches, in which case we wish to find all of them and let the user select between them. It was found that in the test problems, many-to-one mappings⁶⁰ were only desirable when one-to-one mappings did not exist. Therefore when searching for diagram matches, **Dr.Doodle** has an (optional) preference for one-to-one mappings, and will only search for many-to-one mapping when there are no one-to-one mappings. This makes it both easier to use (because the user has to make fewer decisions) and slightly faster⁶¹.

Applying Redraw Rules

This module applies a redraw rule, as specified by definitions 4.3.3.1, 4.3.3.5 and 4.3.3.2. The difficult part of this is creating new objects. Unlike algebraic reasoning, it is not enough to simply introduce a new variable; various parameters have to be set, so that the object can be drawn. This process is partially automated and partially interactive. It is described more in §6.3.3.

Semantic Knowledge

Several aspects of DDLA depend on being able to reliably evaluate implicit relations. This is implemented by a 'semantic knowledge module' which contains functions for testing such relations. In order for DDLA to be sound, these tests must be accurate upto observable differences (c.f. §5.1.1). **Dr.Doodle** does not strictly meet this criterion, at least with the default settings: There are circumstances when the ' $A \subset B$ ' test can return 'true' when in fact A is visibly not inside B . This is due to corners being cut to enhance performance. The ' $A \subset B$ ' test is implemented by testing a sample of points in A for membership of B . In order to strictly meet the requirements of DDLA, it should test a point for each pixel in the

⁶⁰ Where multiple objects in the antecedent diagram are matched with one object in the target diagram.

⁶¹ With one-to-one mappings only, the search space is reduced from $O(m^n)$ to $O(m!/n!)$.

representation of A . Instead, it tests a point for every 8 pixels. Thus if the representation for $A \setminus B$ is narrower or shorter than 8 pixels, there is a chance that the system will not spot it. Note that (a) this corner-cutting can be switched off, and (b) no instances of false answers occurred in any of our experiments.

Performing Implicit Inferences

Implicit inferences are handled using a two stage process: First, equality is handled by substitution. Other implicit inferences are then performed by a Prolog-style first-order horn clause logic prover with loop checking. This is sound and complete for the simple reasoning required.

Detecting Emergent Objects

The emergent object detector spots emergent objects, creates a graphic object to represent them and adds it to the diagram (c.f. §4.4.7). This is implemented using semantic knowledge. It was found to harm system performance in two ways:

- 1) The greater number of objects slows down the diagram matching algorithm.
- 2) More seriously, the greater number of objects leads to many more valid matches when applying rules, which the user has to select between. Most of these are not desirable.

Therefore this functionality is switched off by default. It can be activated via the options menu or by special instructions in tutorial files.

User Interface (UI) Design

This project is not directly concerned with interface issues, and so a detailed description of the GUI - or the design choices involved in making it - would be an unnecessary digression. However, our aim in this project is to create an intuitive easy-to-use logic, so it is clearly important that the system implementing it should also be intuitive and easy-to-use. This means that the user interface is crucial. A considerable amount of work was therefore put into producing a good interface. This section gives an overview of the interface, and discusses some of the design decisions involved in making it. For a hands-on overview, a short introduction to using the program is included on the CD-ROM.

Windows

Dr.Doodle needs to present information in several different contexts and of different types:

- 1) The conjecture being considered

- 2) The working diagram
- 3) The structure of the proof
- 4) The redraw rules available, and tools for applying them
- 5) Instructions/feedback to the user
- 6) Tutorials/lessons

This is too much to clearly present at once. Fortunately only portions of it are needed at any one time. This suggests a system with different windows which can be opened and closed as needed. Certain combinations should be simultaneously visible, and this constrains the size of the different windows.

The system has four main windows: the 'working window' (which handles items 2, 3 and 5 from the list above), the 'theorem window' (item 1), the 'redraw rules window' (items 4 and 5) and the 'lesson window' (items 5 and 6). The different windows are described below. Windows are colour coded to make finding and distinguishing them easy, even when partially covered. The colour-coding uses pastel colours that will not conflict with the use of colour in DDLA (c.f. §4.4.5).

The working window

This window displays the reasoning program (i.e. the structure of the proof) and the currently selected diagram. There is also a message box at the bottom for displaying instructions and comments to the user. A (scaled) screenshot is shown in Figure 6.1.

The reasoning program is a rooted directed acyclic graph (often, but not always, a tree) where nodes represent diagrams. The node corresponding to the currently selected diagram is highlighted with a red border⁶². The tree can be navigated (i.e. different diagrams can be selected) either by clicking on its nodes, or by using the stereo buttons underneath the currently selected diagram.

The diagram viewer allows the user to zoom in – a feature that is especially useful for analysis concepts where tiny differences can be key. A small overview (top-right of the diagram) allows the user to keep track of the whole diagram whilst zooming in on one part. Algebraic statements are drawn in a separate window-pane (right of the diagram), which can

⁶² This should not conflict with the use of colour in DDLA, as the graph/node representation of the reasoning tree is different enough from the DDLA representations as to define a separate context.

be scrolled when too many relations are present to fit cleanly. Right clicking brings up a pop-up menu that lists the object labels. This can be used to select objects when defining a matching (c.f. §6.3.3), or to investigate unclear diagrams (the selected object is highlighted and an algebraic description is given).

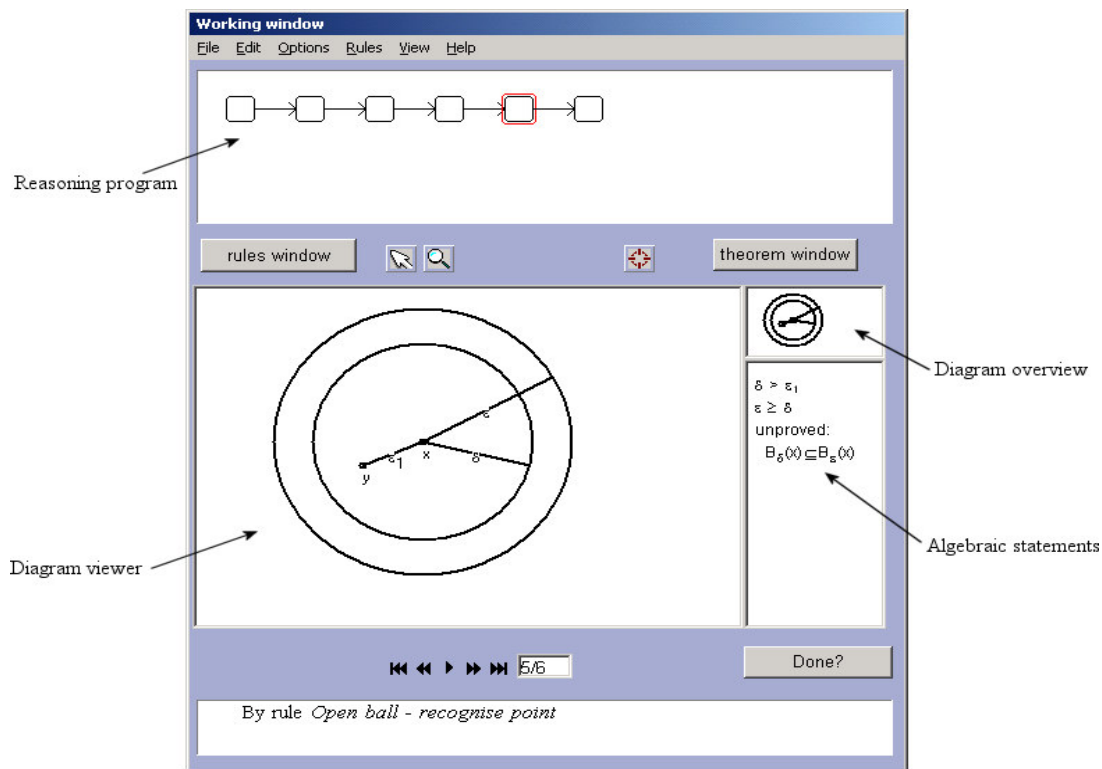


Figure 6.1. The working window showing the reasoning program (top) and the current diagram.

The theorem window

This window (shown in Figure 6.2) displays theorems or rules. Rules are presented diagrammatically and algebraically. Antecedent and consequent are laid out horizontally as

$A \Rightarrow C$, rather than with the $\frac{A}{C}$ convention used in chapter 4 (which, whilst familiar to computer scientists, is less well known amongst mathematicians). A short description of the rule's meaning/use is also given.

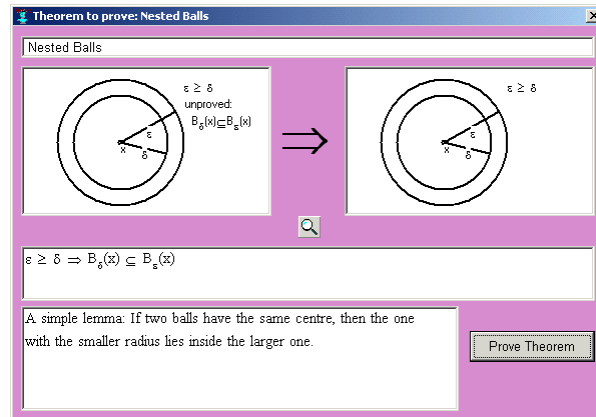


Figure 6.2. The theorem window

For animated rules, video-recorder style controls allow the user to play animations. For branch rules, a 'switch case' button allows the user to flip between the different cases. These controls disappear when not needed – hence their presence alerts the trained user to the type of rule. This representation for branch rules is not ideal, and will probably be changed in future development. It would be better to show both cases simultaneously. This would require more space. The switch case button was implemented to suit the redraw rules window (see below) where screen space is tightly constrained. A better presentation for small spaces might be to show one case as a diagram, with the presence of other cases indicated by node icons as used in the reasoning tree representation. Clicking on a node would switch focus to that case (presented as 'shrinking' the diagram into a node icon, whilst 'expanding' the selected node icon into a diagram).

It is anticipated that theorems will not be as complex – in terms of the number of objects and relations – as proofs. Hence the diagram viewers are smaller than the working window diagram, and do not separate algebraic statements into a scrollable box.

The redraw rules window

This window (shown in Figure 6.3) allows the user to browse, select and apply redraw rules. This window also has a message box at the bottom for displaying instructions or providing feedback. This window must fit onto the screen alongside the working window, hence diagrams are quite small. However the user can also access a more detailed view of the rule (with larger diagrams and a description of the rule's meaning/use). This detailed view uses a copy of the theorem window, but in the colour of the redraw rules window.

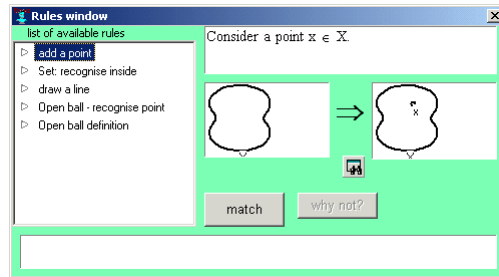


Figure 6.3. The redraw rules window

The lesson browser

Shown in Figure 6.4, this is the top-level window for students. It displays tutorial pages, which can mix text (including mathematical symbols) and graphics. For tutorials introducing the system, it is highly desirable that this window should fit on screen with any of the other windows. Tutorial pages can interact with the rest of the system. Links within the text are used to activate other parts of the system (e.g. loading a theorem, displaying a proof or displaying an example).

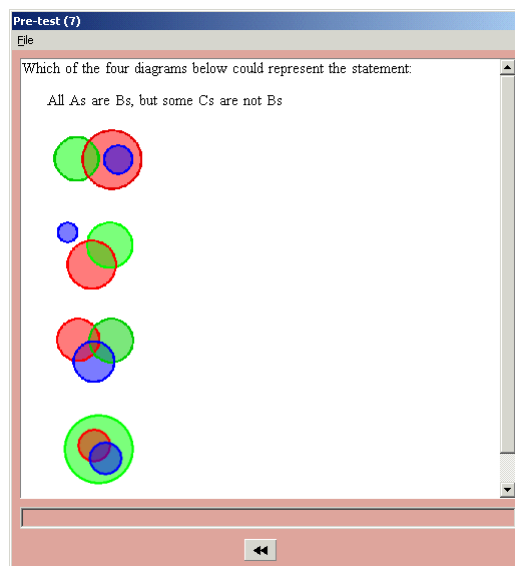


Figure 6.4. The lesson browser

UI - TP interaction

There are several ways in which the top-level program interacts with the underlying theorem prover.

Drawing new objects

New graphic objects require various parameters to be set: a point needs co-ordinates, a ball needs a centre and a radius, etc. These parameters can sometimes be calculated. For example, if the new object is a point y such that $y=f(x)$ and f and x are known. In other cases there are choices to be made, e.g. when drawing an arbitrary point about which nothing is known. The user is then asked to perform a 'drawing action'. The different drawing actions are:

- 1) Picking a point
- 2) Drawing a bounding box
- 3) Specifying a length

All drawing performed by the user is checked by **Dr.Doodle** to ensure that it matches the drawing specified by the rule. Object labels are automatically chosen by the system in a way that is tailored to each object type. They are either selected from lists specific to the object type (e.g. points are x , y , z ,... whilst functions are f , g , h ,...) with subscripts if necessary, or generated from the object's properties (e.g. a ball might be $B_i(x)$, a set might be $X \cap Y$). This approach seems to work well (and the user also has the option of renaming objects).

Rules can, of course, create more than one object at a time. Often, although there are choices involved in drawing several objects, drawing one will fix the parameters of the others. The interactive side of the drawing process is therefore iterative: after each drawing action, the system will attempt to automatically fill in any remaining choices before asking the user again.

Diagram matching

When a rule has multiple possible matches the user must select amongst these. This is done by selecting an object in the antecedent diagram (either by left-clicking on the object, or right-clicking to call up a list of possibilities) and matching it with a suitable object in the current diagram (again, by clicking on an object, or picking from a list). Highlighting and explanatory messages are used to make this easier. This object-object match is then fed as an extra constraint to the matching algorithm, which generates a new, reduced, set of possible matchings. In the test problems, this cycle was repeated at most twice before the matching was fixed.

A different procedure is used in some cases. If the system determines that (a) the chosen rule has several possible matches, (b) the chosen rule creates a new object, and (c) the choice of how to draw this object will determine which match is desired, then the diagram-matching/rule-application process can be streamlined. 'Draw a point' is a good example of a rule where this is often possible. This rule frequently has multiple possible applications (since we could draw a point in any set). However subsequent drawing choices (e.g. where to place the point) usually fix which matching was desired. In such cases, the system does not ask the user to select which match they want. Instead it asks the user to draw the new object (a task they would have to do anyway), then it infers the correct matching from their drawing.

An early version of **Dr.Doodle** experimented with making some common 'basic' redraw rules - such as 'draw a point' - available in the style of drawing tools in a graphics program (i.e. via buttons at the side of the drawing area instead of rules selected in a separate window). This was abandoned in favour of a more uniform approach. The reason for doing so was to make the system easier to learn, which it does in two ways: firstly, a uniform approach streamlines the interface. Secondly, with a uniform approach, basic rules such as 'draw a point' can be used to teach the rule application process for more complex rules. This gives a smoother learning curve.

6.3.4 Algebraic mode

In evaluating this project (c.f. §6.4), we are interested in how DDLA performs in comparison to a traditional algebraic approach – *not* how **Dr.Doodle** performs. To this end, **Dr.Doodle** has been built to work in two modes: diagrammatic and algebraic. This allows us to perform experiments comparing DDLA to an equivalent algebraic logic without the effects of using a computer, the interface design, etc. distorting the results.

In the interests of fairness, considerable effort was put into making the algebraic mode as user-friendly as possible. Where algebra allowed better representations, this has been used. Moreover, 'free rides', which are one of the advantages of diagrammatic reasoning, have been artificially incorporated into the algebraic mode as automated reasoning steps (described below). In terms of presentation, the algebraic mode (which can handle mathematical symbols and subscripts) looks more professional than the diagrammatic mode (whose diagrams are not up to the standards of professional drawing software).

Much of the functionality of the algebraic mode is implemented by converting diagrams into text. Hence a side benefit of the work on this mode is that **Dr.Doodle** can produce an algebraic proof from purely diagrammatic reasoning.

The user interface in algebraic mode

Figure 6.5 shows the working window in the same state as in Figure 6.1, but in algebraic mode.

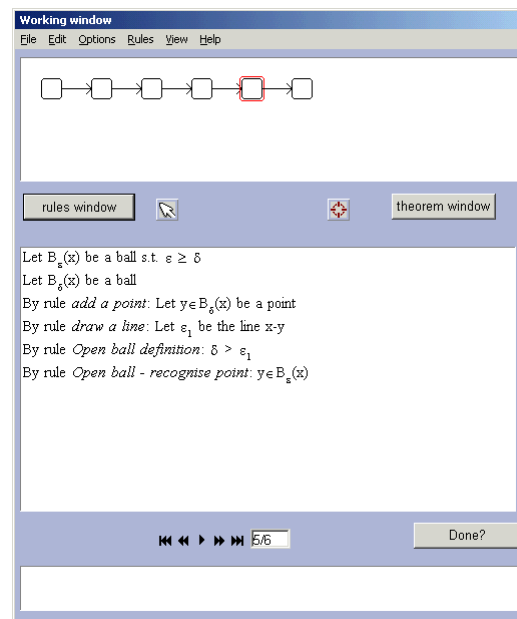


Figure 6.5. The working window in algebraic mode.

The user interface is almost identical to the diagrammatic one. The only differences are:

- 1) The user does not have to draw new objects – their descriptions are automatically created.

- 2) The working window 'page' can optionally display information on the rules used in a proof.
- 3) Animated rules are presented without animation, using quantifier symbols instead.
- 4) Branch rules are presented with all the cases visible at once, instead of needing a switch button to flip between them (i.e. as " $P \Rightarrow$ Case 1: ... *or* Case 2: ...").

Note that except for the use of quantifier symbols instead of animation, all of the interface differences are clearly *in favour* of the algebraic mode. The system can generate algebraic translations of redraw rules. However whilst these are readable, it is often possible to improve on them. For example, the automatically generated ball definition "Given a ball $B_r(x)$, Given a point $y \in B_r(x)$, Let δ be the line $x-y \Rightarrow r > \delta$ " can be simplified to " $y \in B_r(x)$, $\delta = |y-x| \Rightarrow r > \delta$ ". Therefore, for the evaluation experiments, handwritten translations of rules were used.

Reasoning in algebraic mode

The algebraic mode is structurally identical to the diagram mode. That is, it uses the same redraw rule engine for reasoning. However there are of course no implicit relations or emergent objects since these are diagrammatic features. There should be no implicit inferences, as this is also a diagrammatic feature. However they are (optionally) performed as automated reasoning steps. Relations generated by the implicit inference rules are not automatically added to the proof-state (as with diagrams), but such relations are available to the matching algorithm. When an implicit inference is used, this is stated in the proof state, as shown in Figure 6.6. This is done so that algebraic proofs are as short as diagrammatic ones and do not involve more rules.

```

    Let  $\varepsilon_1$  be the line x-y
    By rule Open ball definition:
       $\delta > \varepsilon_1$ 
    By rule Open ball - recognise point (using  $\varepsilon \geq \delta, \delta > \varepsilon_1$ ):
       $y \in B_\varepsilon(x)$ 

```

Figure 6.6. A proof-in-progress in algebraic mode, showing an automated inference.

The rule 'Open ball - recognise point' requires $\varepsilon > \varepsilon_1$, which is automatically deduced from $\varepsilon \geq \delta, \delta > \varepsilon_1$.

6.3.5 Testing

Dr.Doodle was tested both at the level of individual functions and as a whole system, to verify that it behaves as required (i.e. according to the rules of DDLA). These tests involved a wide range of problems, but were not in any way exhaustive. **Dr.Doodle** was also tested during the experiments described in §6.4. This involved only a small set of problems, but with a number of users (who – through working in different ways – provide a better safeguard than a single tester). Again, these tests showed that **Dr.Doodle** behaves as required. There are bugs in **Dr.Doodle**. However these lead to system failure, rather than unwanted behaviour. Whilst this is not desirable, it does not reflect on our claim that DDLA can be performed by a computer. There was no formal verification of the code (which would be highly unusual for such a project). Given the size and complexity of the **Dr.Doodle** program, it is highly likely that there *are* some bugs that produce unwanted system behaviour. We consider it a sufficient standard of correctness for a project such as this, that no such bugs were found on a reasonable range of test problems.⁶³

6.3.6 Summary

In order to evaluate DDLA – both as a valid formal logic and as a more intuitive alternative to conventional logic – we have produced a computer implementation which we call **Dr.Doodle**. **Dr.Doodle** is an interactive theorem prover which can perform both diagrammatic reasoning and algebraic reasoning. There are some differences between DDLA and **Dr.Doodle** (c.f. §6.3.2). We judge that these differences are minor, and do not cast doubt on the fact that DDLA reasoning can be carried out by a computer.

⁶³ A pessimistic view: this just demonstrates the inadequacy of the tests.

6.4 Empirical evaluation

In our hypothesis, the hardest part to test is the criterion regarding human usage. There are various possible experiments that could be carried out:

1. Test for user recall of definitions and proofs.
2. Test for user production of new proofs to similar but previously unseen problems.
3. Collect anecdotal evidence on users impressions.
4. Analyse the amount of searching done by users in producing a proof.
5. Test how difficult it is to spot that a theorem is beyond the system's capability to prove or disprove.
6. Test for the effect of using the interactive theorem prover on subsequent pen-and-paper work.

It would be interesting to carry out these tests for both experts and non-experts in the domain. In practice it was not possible to carry out all these tests, and we focused on non-experts, since the system is conceived of as an educational tool. Also, visual representations are likely to be most beneficial to non-experts, as experts may have already developed their own visualisations.

6.4.1 Constraints

The main constraint on the design of the experiments was the difficulty of finding suitable volunteers. It was decided that the experiments must be kept relatively short, so that the time commitment required from volunteers was not prohibitive. We fixed on 1½ hours as a reasonable compromise between attracting volunteers and experiment content.⁶⁴ This limited both the material that could be taught and the testing that could be done of student's learning. Other constraints were the need to extract data suitable for objective analysis, and the tight time-frame for this stage of the project.

6.4.2 Test subjects

We tested two groups of subjects:

1. First year undergraduate mathematics students.

These (hopefully) have some mathematical ability, but were unfamiliar with the domain, and probably only have a loose idea of the nature of mathematical proof. They were tested in their 2nd term, before they covered any analysis concepts in their course. In

⁶⁴ By comparison, Stenning *et al*'s studies on HYPERPROOF were conducted over the length of a full course in logic.

particular this means that they were unlikely to have formed their own internal representations for such concepts.

2. Older (2nd and 3rd year) undergraduate mathematics students.

These have studied analysis, and may have developed their own internal representations for concepts from the domain. They should also be more familiar with the nature of mathematical proof.

Experiment Design

First Year Undergraduates

We wished to compare behaviour between diagrammatic reasoning and equivalent algebraic reasoning. Thus the subjects were randomly divided into two groups: one worked using diagrams, one using algebra. We wanted to present the diagrams and the algebra in as similar a form as possible in order to prevent other effects interfering with the results. To this end, **Dr.Doodle** was built to work in two modes (diagrammatic and algebraic). As discussed in §6.3.4, the two modes are structurally identical. This should minimise the potential effect of differences in the interface/presentation methods. Both diagram and algebra groups were taught identical material (although the wording of the teaching material differed slightly in places). The experiments were conducted in groups of between 1 to 3 students. Each student had a computer running the **Dr.Doodle** system. Cooperation was not allowed. I provided support for problems in using the system, but not mathematical problems.

1st session: Training

Students were introduced to the system and taught the concepts of *open sets* and *continuous functions*. The timetable for these sessions was:

- 5 minutes spoken introduction.
- 5 minutes pre-test of basic spatial and algebraic abilities.
- 30 minutes working with the system.

2nd session: Testing

These sessions took place between one and two days after the 1st session, allowing time for the material to be absorbed but hopefully not forgotten. The timetable was:

- 5 minutes re-familiarising students with the system through simple tasks.

- 30 minutes solving problems of increasing difficulty. These problems were all of the form “Given P , prove Q ”. For each problem, students were given the correct set of rules needed to solve it.
- 5 minutes filling in a feedback form.

Data gathered

The experiment collected the following data for analysis:

- 1) Crude measures of basic spatial and algebraic abilities via the pre-tests.
- 2) Success at solving the test problems.
- 3) A log of the following user actions (with time of action).
- 4) Student's views via the feedback forms. These asked students to rate the system out of 5 on the following criteria:
 - a) Easy to learn
 - b) Easy to use
 - c) Helpful for learning mathematics
 - d) Helpful for understanding mathematics
 - e) Helpful for doing mathematics
 - f) Overall score

They were also asked for comments on each rating.

Older Undergraduates

Both the teaching materials and the **Dr.Doodle** program were revised slightly in the light of the first set of experiments (c.f. §6.4.3 below). The changes made were:

- 1) Experiments were conducted in one sitting of 1½ hours with a five minute break half-way, instead of two sittings of 40 minutes. This change was made because in the previous experiment:
 - a) Some of the students dropped out after the first session.
 - b) When asked what would make volunteering more attractive, some of the students commented that they would have preferred a single session.
- 2) Some minor improvements in **Dr.Doodle** (both to the interface and in terms of stability)
- 3) Only one analysis concept – open sets – was taught.
- 4) The number of exercises on open sets was increased from 3 to 13. This was done so that question score would give a better (finer grained) measure of performance. The new exercises were not all of the same type. As well as problems of the form “Given P prove

Q ”, students also had to answer true/false questions (i.e. “Is $P \Rightarrow Q$ a theorem?”) and a counter-example question.

Otherwise the experiments were identical.

6.4.3 Results

First Year Undergraduates

10 subjects were recruited.⁶⁵ They were randomly assigned to the diagrams or algebra test group so that each test group had 5 subjects. The data gathered from this experiment tentatively supports our hypothesis.

Teaching both open sets and continuity turned out to be too ambitious. Only a couple of the students reached the exercises on continuity, and none completed any of them. Hence our results are limited to only three problems on open sets. This, combined with the small sample size, makes success or failure on the problems too coarse grained a measure: the results from the two groups are virtually identical. The students had mixed success: some got all 3 solutions out, some gave up, some ran out of time. The average success rate was exactly 60% for both groups.

Data from the feedback forms is unreliable. The presence of the experimenter and the politeness/timidity of the 1st years combined to encourage only positive responses from most of the students. The average score for each question was 4 out of 5. In my opinion, this did not reflect their 'true feelings', as most of the students were (understandably) quite lost for much of the time. The comments were similarly too positive to be trustworthy as evidence. The diagrammatic-reasoning group mostly made the right sounds:

“It helps to show maths 'in action'”

“The pictures were useful for helping understand what was going on. Better than written explanations a lot of the time.”

“Easier to understand if things are presented visually which would make this a very useful learning aid, as it would be remembered more easily”

⁶⁵ 20 students were signed up, but only 10 turned up.

Although there was one negative comment:

“While some things are easily visualised as pictures, I find others get too complicated and written stuff is better.”

The user logs provide more interesting results. For each successfully completed problem, we calculate two measures of 'unnecessary actions' as follows:

- 1) Each attempt to apply a rule counts as a 'move'
score = (no. moves made)/(minimum no. moves necessary)
- 2) Each attempt to apply a rule and each navigational step counts as a 'move'
score = (no. moves made)/(minimum no. moves necessary)

These scores were only calculated for complete answers. We could also include incomplete proofs, however it is unclear how these should be compared with those from complete answers. Since the completion rate was identical for both groups, this should not distort the results. From individual scores, we calculate the mean unnecessary action score per question for each group. The results are shown in Table 1. They show the diagrams group making considerably fewer mistakes. Unfortunately the small sample size means that we cannot draw any firm conclusions from this experiment.

	Measure 1	Measure 2
Diagrams Group	1.49, $\sigma=0.98$	2.29, $\sigma=0.95$
Algebra Group	2.74, $\sigma=1.99$	3.32, $\sigma=1.81$

Table 1. Results from experiments with 1st years.

Older undergraduates

10 subjects were recruited. They were randomly assigned to the diagrams or algebra test group so that each test group had 5 subjects. One run was rejected (the student turned out to be a wayward biology student), and so another was randomly rejected from the opposing group, leaving 8 students. However in spite of the small size, the data gathered from this experiment showed statistically significant support for our hypothesis.

The changes in the experiment design were successful in producing richer results. The average score (percentage of exercises correctly answered/solved) for the diagrams group was 63% (with a standard deviation of 6%), whilst the average score for the algebra group was only 42% (with a standard deviation of 20%). Moreover this difference *is* significant

(with 95% confidence using a one-tailed t -test). Hence the experiment supports our hypothesis.

As in the previous experiment, we also see the diagrams group solving problems more quickly (i.e. using fewer interactions with the system, as measured by the two scores used for judging the 1st years' performance). The 'Measure 1' results here are statistically significant (with 95% confidence using a one-tailed t -test).

	Score	Measure 1	Measure 2
Diagrams Group	63%, $\sigma=6\%$	1.27, $\sigma=0.03$	2.12, $\sigma=1.08$
Algebra Group	42%, $\sigma=20\%$	1.63, $\sigma=0.19$	2.41, $\sigma=0.63$

Table 2. Results from experiments with 2nd/3rd years.

Again, the pre-test was a poor indicator of performance. There was some correlation between displaying a bias towards sentential or visual reasoning in the pre-test, and subsequent performance on the exercises. However the effect is too weak for any conclusions to be drawn.

The data from the feedback forms is less positive than from the 1st years, and therefore perhaps more reliable. Perhaps surprisingly, the algebra group were slightly more positive. They gave the system an overall score of 3.75/5, versus 3.25/5 from the diagrams group. This pattern of a slightly higher score from the algebra group was repeated in the more specific questions, with the exception of the question on understanding where the diagrams group gave a slightly higher score (3.5 versus 3.25). This pattern can be at least partly explained by differences in presentation standard: the algebraic mode of **Dr.Doodle** looks and feels more 'professional'.

The subjects were much more sparing in their comments than in the previous experiment. Most did not write anything, and the few comments we did get were almost entirely related to minor defects in the system's interface. There was only one comment relevant to the use of diagrams: "I found it very useful to visualise the problems".

6.4.4 Future work

Whilst we have found statistically significant results, our evaluation of using DDLA is by no means exhaustive. Further experiments are desirable. Most obviously, the experiment conducted could be repeated (a) with more subjects, and (b) looking at usage over a longer

time period (allowing us to both cover more concepts and investigate learning rates). Different experiments are also desirable. In §6.4 we identify 6 ways in which the usage of two logics could be compared:

- 1) Compare recall of definitions and proofs.
- 2) Compare the production of new proofs to similar but previously unseen problems.
- 3) Anecdotal evidence of user's impressions.
- 4) Analyse the amount of searching done by users in producing a proof.
- 5) Compare how difficult it is to spot that a theorem is beyond the system's capability to prove or disprove.
- 6) Test for the effects (if any) of using the interactive theorem prover on pen-and-paper work.

Of these, we have so far only carried out (2) and (3). There is clearly room for several further studies on the question of the advantages and dis-advantages of systems such as DDLA/Dr.Doodle. And that is without touching on the question of *why* the system gives these advantages and dis-advantages. It would also be interesting to test some of the design ideas: For example, what difference does animation make? What difference does altering the implicit relation rules or emergent objects make?

6.4.5 Conclusion

These experiments could be criticised on several grounds, largely related to scale: they should have involved more students, covered more material and subjected the students to more tests. In particular the 5-minute pre-test to determine sentential/visual bias was too crude to be useful. Unfortunately, a more thorough empirical investigation was beyond the scope of this project.

However these experiments do loosely support the hypothesis that diagrammatic reasoning is – in some sense – easier than algebraic reasoning in this domain. The performance measures show those students using diagrammatic reasoning outperforming those using algebra. Although many of our results are not statistically strong, probably due to the small size of the test groups, we did find a statistically significant difference between the two reasoning styles for the older group.

6.5 Summary

This project set out with the aims of producing a formal logic that uses diagrammatic reasoning to solve problems in real analysis, and demonstrating that this was better in some way than conventional logic. Considering this, we fixed on the hypothesis:

"Diagrammatic proofs are possible for analysis problems, and may be easier (in some sense) than algebraic proofs"

This hypothesis was evaluated against four criteria:

- 1) Soundness.
- 2) Range.
- 3) The reasoning can be implemented on a computer.
- 4) The proofs are 'easier' for a significant proportion of people.

DDLA passes the soundness (with acceptable qualifications). Its performance is mediocre on range, covering a fair number of interesting theorems, but missing many important concepts and theorems. It seems plausible that the limited success on this criterion is due to the limited scope of this project, rather than any inherent limitations in diagrammatic reasoning (see §8.2 for suggestions to extend DDLA). Hence we judge that mediocre performance here is sufficient, and is not evidence against the hypothesis.

DDLA passes the implementation criterion.

Our testing of criterion (4) was limited. The results we have are positive, but further experiments would be required to draw any definite conclusions.

Based on this analysis, we conclude that DDLA successfully meets its aims. Although not in any way conclusive, this gives a reasonable case in favour of the hypothesis.

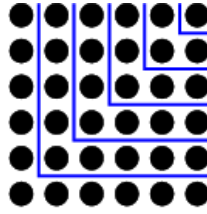
7 Related Work

In considering how DDLA/*Dr.Doodle* fits into the field, we examine the most closely related work. Diagrammatic reasoning is a relatively new area of research, and there is little directly related work (c.f. §2.3). Moreover, this project has focused on a domain that has not previously been examined by the diagrammatic reasoning community. Nevertheless, we can find similarities between our work and some prior (and ongoing) research projects. In §2.3, we identified two such systems: *DIAMOND* and the constraint/spider diagram representation.

The 'dynamic' style of reasoning that we develop (i.e. reasoning by drawing, rather than reasoning from pre-drawn diagrams) shares ideas with Jamnik's *DIAMOND* system [24]. This is not surprising, given that this project began as an extension of *DIAMOND*. The representation used for objects and relations in DDLA owes much to common conventions (e.g. the use of arrows to denote functions). The representation we have developed for logical concepts (such as quantifiers) has novel aspects, but comparisons are possible with other systems. Of particular interest is Howse *et al*'s work on constraint/spider diagrams, which covers quantifiers [11] (and is perhaps currently the only 'live' project in the field that does so).

Our comparison shows that all three systems (DDLA, *DIAMOND* & spider diagrams) are quite different, and none are well-suited to the other's field. We trace these differences to differences in the domain and the goals of each system. This highlights some of the design principles underlying their development. Whilst ideas from one area can be applied to another, it seems highly unlikely that a universally superior representation system exists.

7.1 DIAMOND



$$1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

Figure 7.1. A DIAMOND-style proof.

Jamnik's DIAMOND system uses diagrams to prove theorems in natural number arithmetic. Diagrams in DIAMOND consist of arrangements of dots. The dots represent numbers (i.e. 3 dots for the number 3), and their arrangement represents relations between them (e.g. a square arrangement for the relation $9=3^2$). A proof consists of constructing a number in one way (e.g. drawing a square to get a square number), then manipulating the diagram – often by deconstructing it – to show that the number has other properties (e.g. breaking a square into nested L-shapes to show that $3^2=5+3+1$). The DIAMOND system supplies tools for performing these drawing steps and manipulations. It requires the user to supply example proofs for a couple of specific cases. The system then attempts to find a general proof procedure which will work for all cases. The validity of the general proof procedure is checked using meta-induction (that is, by showing that if $P(n)$ is a valid proof for case n , then $P(n+1)$ is a valid proof for case $n+1$).

7.1.1 Similarities

This project was initially conceived as extending Jamnik's work to a continuous domain, and so it is not surprising that there are some strong similarities between the two systems:

- Both systems use schematic proofs (i.e. proofs as programs which generate a family of specific-case proofs) defined using specific cases.
- The drawing/manipulation steps in DIAMOND can be thought of as redraw rules.
- Both DDLA and DIAMOND use rectangles ('area blocks' in DDLA) to represent multiplication (e.g. $y=x^2$) as an implicit relation.

7.1.2 Differences

Although this project took Jamnik's work as its starting point, the differences between the domains – one countable, the other continuous – have led to radical differences in the final reasoning systems.

Differences in reasoning methodology

- Although DDLA uses specific examples for representations, it uses reasoning steps that can be applied to any case and so produces one proof for all cases. By contrast, in DIAMOND the specific proofs are altered for each case. This is because for theorems provable in DDLA, there are smooth transitions between different cases, whereas in DIAMOND's domain there are discrete jumps between cases. Thus the range of theorems that the two systems can prove is entirely distinct, and this is not simply because each has not been applied to the other's domain.
- DIAMOND does not cover counter-example reasoning. It is not clear that diagrams are particularly useful for counter-example reasoning in DIAMOND's domain.

Differences in representation

- Quantifiers: DIAMOND only handles concepts of the form $\forall n.p(n)$. DDLA has to handle concepts involving more complex quantification such as alternating quantifiers.
- Objects: The two systems handle disjoint collections of objects. DIAMOND represents natural numbers. DDLA can represent a range of different objects (sets, real numbers, functions, etc.) – but not natural numbers.
- Relations: DIAMOND can state relations connected to object shape (e.g. $a=b^2$). DDLA incorporates a variety of ways of stating relations: implicitly (e.g. $x \in X$), graphically (e.g. $\text{open}(X)$ or $a=f(b)$) or algebraically (e.g. $\text{cts}(f)$). This allows it to cover a wider range of relations.
- Directness: All DIAMOND's representations are direct, whilst some DDLA representations are indirect (e.g. 2D functions). This is a necessary consequence of DDLA's domain.
- Clarity: Clarity is not an issue in DIAMOND, as its domain allows for the clear separation of diagram objects. Hence, reasoning in DIAMOND does not require the assumption made in §5.1.1 that users will create readable diagrams.

Other Differences

There are also differences that stem from the different goals of the two projects. The work on DIAMOND was motivated by theoretical concerns (i.e. demonstrating that the type of reasoning it embodies can be formalised). The system was not intended to be a practical tool, and consequently its potential applications are limited. It is a proof checker for established proofs that humans find intuitively clear and do not need checking. Since the user must also supply the proof, it is hard to see any applications, at least without further work. By contrast, this project had both theoretical and practical goals. As a result, its design was geared towards producing a practical and user-friendly system with educational applications in mind.

7.2 Spider diagrams & constraint diagrams

The spider/constraint diagram family of representation schemes is perhaps the most successful work on representing disjunctions and negations in diagrams. Spider diagrams are an extension of Euler circles and Pierce Diagrams developed by Howse *et al* [22]. Spider diagrams can express membership and subset relations and cardinality constraints. Reasoning systems have been developed for spider diagrams, and been shown to be sound. The objects in spider diagrams are zones formed by oval contours (representing sets with overlap=intersection), points (representing members of sets), plus relation representations such as 'spiders' (linked points representing statements of the form $x \in A \cup B$), ties, strands and shading (allowing cardinality statements). Spider diagrams are not only more powerful than Euler circles, they are also easier to use, due to innovations such as *projections* (c.f. [12]) and (obviously) *spiders*. Constraint diagrams further extend spider diagrams by allowing universal quantification and relational navigation (c.f. [9]). Constraint diagrams include arrows (representing arbitrary relationships/functions) and compound diagrams (a disjunction of diagrams). Figure 7.2 shows an example spider diagram, and Figure 7.3 an example constraint diagram.

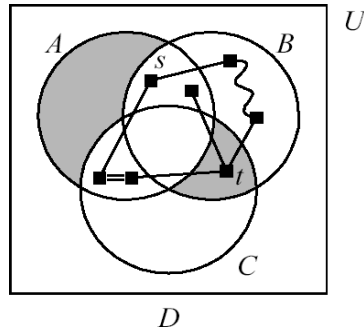


Figure 7.2. A spider diagram.

Constraint/spider diagrams are a very different system to DDLA. A short list of the differences serves to show that the two systems cannot be considered rivals:

- Although reasoning methods have been developed for constraint diagrams (now including automated reasoning), their primary use is as a representation system – whilst DDLA is primarily designed for reasoning. The reasoning methods used in the two systems are too different for a fruitful comparison to be made.
- Constraint diagrams are general purpose, whilst DDLA is domain-specific.

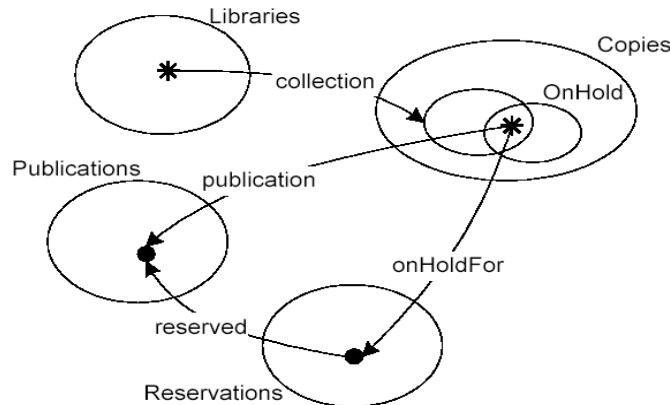


Figure 7.3. A constraint diagram expressing (amongst other constraints) “Any library book that is on hold, must be the same publication as that associated with the reservation for which it is on hold.”

- Constraint diagrams use an abstract indirect representation (e.g. within a single diagram, points can represent people, books, etc.), whilst DDLA objects are as direct as possible (i.e. tied to specific (classes of) models).

Nevertheless there are interesting comparisons that can be made between constraint diagrams and DDLA diagrams. Both tackle the issues involved in representing complex statements involving quantifiers and arbitrary relations. The solutions in [11] for handling quantification have similarities to our approach. It is also instructive to compare how the two systems handle disjunction and negative statements.

7.2.1 Emergent objects

Emergent objects occur in spider diagrams when a *zone* is created by the intersection of two other zones. Spider diagrams make use of these emergent zones. The same situation can arise in DDLA – however after experimenting with emergent objects, we decided instead to require that the user must explicitly create intersections if they want to use them (c.f. §4.4.7).

For representation purposes, the spider diagram interpretation is more intuitive. There are two reasons, though, why it does not work well for DDLA:

- 1) Without spiders (which can be used to ignore emergent intersections), a diagram with emergent objects often requires more *unknown* relations. This distracts attention from more important relations.
- 2) When applying rules, the greater number of sets represented in the diagram leads to a greater number of valid matches, which the user then has to select between. Most of these are not desirable, and so having emergent objects slows down the rule application process.

The first problem is a consequence of a design choice regarding representing disjunction, which we could make differently (c.f. §7.2.2). However the second problem cannot be avoided in the type of rule-based reasoning we use. Hence this difference reflects the different intended uses of the two representations.

7.2.2 Representing disjunction

DDLA handles disjunction by giving multiple diagrams. Spider diagrams do so by introducing a special notation – the *spider*. Spiders allow several cases to be shown in one diagram. Although arguably not as straightforward as using different diagrams for different cases, spiders are reasonably intuitive once learnt. Moreover, they avoid splitting into multiple diagrams. This is a strong asset, since it allows for a much more compact representation.

This disjunction notation only works for the membership relation. Membership statements combined with appropriately labelled sets can be used to represent disjunctions involving other relations (e.g. “ $x < y$ or $x \geq y$ ” can be represented as “ $x \in \{x': x' < y\} \cup \{x': x' \geq y\}$ ”). However this forces us to represent relations in terms of sets, which does not seem to be an intuitive representation for all relations. Other conventions can be used for disjunctions involving other relations (e.g. the use of '?' shapes in HyperProof – c.f. §3.3.1). The disadvantages of doing so are:

- 1) Each such convention must be learnt by users, creating a steeper learning curve.
- 2) More importantly, developing a set of conventions that gives good representations for disjunctions involving a range of different relations is a hard, and perhaps impossible, project.

Since we need to represent case splits over a variety of relations in our work, the specialised notation approach embodied by spiders seems unsuitable. The multiple diagram approach is not ideal, but it is the simplest and most flexible. Hybrid approaches – using spiders for disjunction over membership (i.e. $x \in P \cup Q$) – and multiple diagrams for other disjunctions – are possible. A good case can be made for giving the inside relation special treatment, since it is such a strong visual representation. Hence a hybrid approach would probably be better, even though it would involve slightly more learning for users. In future developments of DDLA, we will strongly consider adding spiders.

7.2.3 Representing negatives

Constraint diagrams can represent negative statements of the forms $a \neq b$, $a \notin A$. These statements are represented as implicit relations (that is by drawing two separate points for $a \neq b$, and by drawing a outside A for $a \notin A$). Explicit annotations can be used to over-ride this default reading.

DDLA does not allow implicit negative relations.⁶⁶ Not only does this mean that DDLA must rely on algebraic statements for negation, it is also probably less intuitive than the spider diagram interpretation (at least when learning the system). The reason we do not use implicit negative relations in DDLA is that we usually wish to reason about the general case (e.g. $a = b$ or $a \neq b$; typically $a = b$ will be a 'degenerate' case), yet have the proof cover all

⁶⁶ Although $a \notin A$ can be represented implicitly, provided A^c is explicitly created before drawing a .

cases. Hence implicit negative relations would require almost constant use of the explicit over-ride annotations.⁶⁷ Thus this difference can be traced to the different primary uses of DDLA and constraint diagrams (i.e. reasoning versus representation).

7.2.4 Representing implication

Spider diagrams can represent statements of the form $P(x) \Rightarrow Q(x)$ using sets (i.e. by drawing $\{x:P(x)\} \subset \{x:Q(x)\}$). DDLA uses redraw rules with an antecedent and a consequent instead.⁶⁸ Both are equally general. However the set based representation is arguably more natural for taxonomy-type statements (e.g. “ $\text{penguin}(x) \Rightarrow \text{bird}(x)$ ”), whilst the rule based representation is better for other statements, especially those involving non-unary relations (e.g. “ $\text{loves}(x,y) \wedge \text{loves}(y,x) \Rightarrow \text{happy}(x)$ ”).

7.2.5 Representing quantifiers

Only points and spiders are quantified over in constraint diagrams. Quantifier type is handled by drawing points in different ways. That is, the representation uses different primitive objects for the different quantifiers: \bullet for “ \exists a point”, \star for “ \forall points”. Using different objects to represent quantifier type would not be suitable for our analysis diagrams, where we variously wish to quantify over points, sets, functions, lengths, etc., as it would involve introducing multiple representations for each type of object. This would quickly get confusing. Using two colours to distinguish between quantifier types is equivalent though, and adapts well to multiple object types.

Quantifier ordering problems are dealt with by labelling quantifiers with numbers. This is logically equivalent to our use of animation. It lacks the potential appeal of linking hierarchy to chronology, but is better suited to 'static' mediums (e.g. paper representations), where animation is reduced to comic-book style diagram chains which both consume more space and are harder to follow.

As noted in §4.2.7, DDLA only expresses one nesting of quantifiers (i.e. questions of quantifier scope are settled by restriction). Constraint diagrams allow more complex scoping of quantifiers (i.e. multiple nestings) provided the objects involved are linked by relation arrows. A quantifier's scope is determined by analysing the dependencies of the objects

⁶⁷ Using appealing graphical annotations (e.g. spiders) would improve this somewhat.

⁶⁸ Using subset for implication is possible in DDLA, but it is not something we have explored.

involved. It should be possible to apply a similar strategy in DDLA, although we have not found it necessary so far. See [9] for details of the constraint diagram approach.

7.2.6 Domain flexibility

Constraint diagrams can represent statements from any theory based in first-order set-theory with functions. This is a very flexible framework that can be applied to a wide range of domains. By contrast, the semantics of DDLA specify that objects are interpreted as being in \mathbb{R}^2 , and hence DDLA can only be applied to domains that contain \mathbb{R}^2 . This is a key difference between the two systems, from which many of the other differences stem: spider diagrams grew out of work on UML, and are intended to be useful in a range of disparate domains; DDLA is specialised for abstract geometry.

The two systems illustrate well the trade-off between flexibility and 'power'. DDLA cannot be used except for analysis (and, potentially, related subjects), but constraint diagrams – whilst in principle capable of working in any domain – are not best suited to geometric domains. This is demonstrated by trying to represent DDLA rules as constraint diagrams. Figure 7.1 shows the same statement, represented in constraint diagrams and as a DDLA rule. By using specialised object types and specialised ways of stating relations, DDLA gives a much simpler and clearer representation.

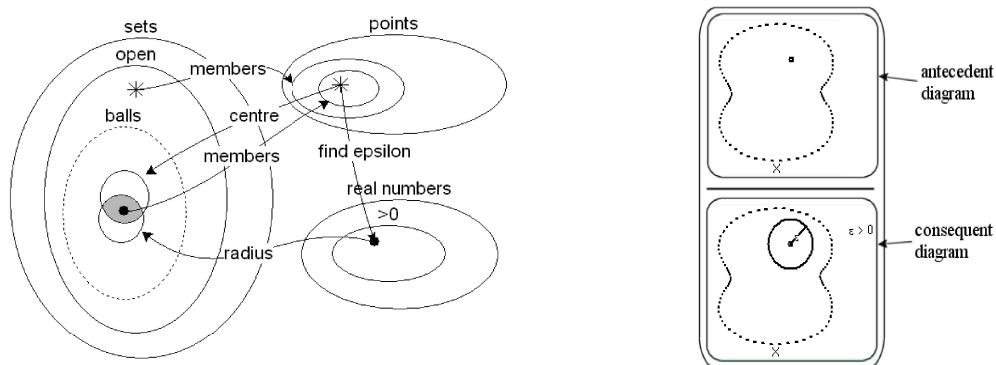


Figure 7.4. Expressing "If X is an open set..." using a constraint diagram (left) and a DDLA rule (right).

7.3 Summary

We have compared DDLA with what we judge to be the most similar related research. This has shown links between DDLA and other work in the field, but also large differences.

Reasoning in DDLA has some similarities with reasoning in Jamnik's DIAMOND system. Both use a 'dynamic' style of reasoning (i.e. reasoning by drawing/transforming diagrams, rather than reasoning from pre-drawn diagrams), and work with specific cases whilst proving general theorems. However, because of the deep differences in their domains (DIAMOND works in a discrete domain, whilst DDLA operates in a continuous domain), the reasoning style is very different.

The representation we have developed for logical concepts (such as quantifiers) has novel aspects, but comparisons are possible with other systems. Of particular interest is Howse *et al*'s work on constraint/spider diagrams, which covers quantifiers [11] (and is perhaps currently the only 'live' project in the field that does so). Constraint diagrams are a powerful diagrammatic representation system. Our comparison shows that the two systems (DDLA & spider diagrams) are quite different. We can trace many of these differences to differences in the domain and goals of each system. Hence this comparison sheds light on some of the principles underlying certain design choices.

We conclude that none of these systems can be considered to be 'rivals'. Whilst ideas from one area can be applied to another, none of the three systems considered here are well-suited to the others' fields. It seems that diagrammatic systems are inherently prone towards some degree of specialisation, and it is therefore unlikely that a universally superior representation system exists.

8 Future work

Future work can be divided into extensions to the **Dr.Doodle** system (i.e. improvements to usability), and extensions to DDLA (i.e. increasing the range and power of the logic). There may also be more work that can be done with the system as it is, since we have not definitively explored its limits.

There are several ways in which **Dr.Doodle** could potentially be improved (including automated drawing, automated reasoning and interface improvements). Some of these are straightforward extensions, some developments requiring theoretical work. Future work here also includes investigating how the different features of the system are used in practice, and what effect different settings/configurations have on users. The most important line of further work is in extending the range of the logic. We look at several ways in which DDLA could be extended. There are many concepts of interest in the domain that we have not covered in this project. These include connected and disconnected sets, differentiable and integrable functions, metrics and contraction mappings. Some of these could be covered within the existing framework by designing suitable representations and adding definitions. However for some we would need to extend the reasoning mechanism. Perhaps the key extension is sequence reasoning, which would greatly extend the power of the logic.

In extending DDLA, we have to be careful not to make it too complex. Any amount of information can be represented and manipulated in a diagram, but we risk losing the intuitive appeal of diagrammatic reasoning. There seems little point in developing a diagrammatic logic as powerful as sentential logic if it is also as hard to follow. As described in §6.4.4, further studies of how the logic is used are also important.

8.1 Improving Dr.Doodle

8.1.1 Improving the user interface

User interfaces take a lot of work to perfect, and no doubt there are many ways in which the current interface could be improved. One clear improvement would be the addition of context sensitive help. It would also be beneficial to provide informative feedback when an action fails (e.g. “Quantifier error: x is not an arbitrary point”). This involves guessing what the user wanted (i.e. which matching the user had in mind when attempting to apply a rule). This could be done by ranking flawed matchings using a measure for 'goodness of fit' based on the number and type of broken constraints. Calculating such feedback could be computationally expensive (as we could not prune the matching search space as aggressively), but would only need to be done on demand. Another possible improvement would be to allow users to draw diagram objects in a freehand manner (with the mouse or a graphics tablet), with the system automatically recognising the objects drawn, finding the appropriate rule for creating them, and testing the validity of such a proof step. This would be technically ambitious, but not theoretically difficult.

8.1.2 Representation

Most obviously, the standard of the diagram-drawing routines could be improved. Embedding a third-party diagram viewer would be the most desirable option here, if a suitable one could be found.

The current display methods are biased towards displaying diagrams without much algebra. A more flexible display method that supports more mixing of algebra and diagrams is desirable. A fairly simple adaptation would be to make the representation for the current proof state (currently 3 window panes: a large diagram, a list of algebraic statements & a small overview diagram for zooming) consist of a flexible number of resizable window panes. This would also allow conjunctions of diagrams, which might be useful in complex proofs.

It would be interesting to investigate the use of context sensitive representations for rules. For example, rules such as `add-a-set` could be drawn with 1D sets when working with 1D spaces.

Changeable implicit relation rules

As noted in §4.4.5, the set of implicit relation rules used in DDLA is not rigidly determined, but is instead a design choice tailored to the domain. **Dr.Doodle** could allow users to change which relations are represented implicitly.⁶⁹ Potentially, the computer system could decide based on context which implicit relation rules to use, but this could easily be confusing to users.

8.1.3 Full DDLA compliance

As noted in §6.3.2, **Dr.Doodle** does not completely implement DDLA. It could be extended to do so, which would allow us to investigate how useful flexible-matching/condition-updating is in practice. This requires more coding, but no significant unresolved technical issues.

8.1.4 Automated drawing

At present, whilst the system calculates how to draw those objects whose parameters are fixed (e.g. y where $y=f(x)$ and f, x are already drawn), all drawing choices are made by the user. This is probably the behaviour expected by the user, but it is occasionally intrusive. For example, when performing a case split, it is up to the user to modify the diagram to give new cases, which can be confusing for novice users.⁷⁰ Also, automated diagram drawing is necessary if we wish to investigate automated diagrammatic reasoning.

There are two ways in which we might increase the degree of automation in the drawing process. One is to develop heuristics for drawing 'good' diagrams. Diagrams should be as *accurate* as possible (so that as many of the relations are true as possible, noting that this may not be all relations), whilst avoiding unintentional implicit relations, and being 'sensible' (i.e. clear to the user). This is a hard and open-ended problem.

I suggest the following 'proto-algorithm'⁷¹:

- The drawing algorithm should work forward, with all drawing choices about an object being made at the point where that object is first drawn. This should greatly reduce the

⁶⁹ Note that implicit relations must be testable in a decidable fashion, e.g. relations such as ' $x < y$ ' are suitable, but not ' $\text{open}(X)$ '.

⁷⁰ The most recent version of **Dr.Doodle** does attempt to automate case-split redrawing for some simple cases.

⁷¹ 'proto-algorithm' is used here as shorthand for 'extensive research programme'.

complexity of the problem, as typically a reasoning step will introduce at most a couple of new objects. It has the drawback that initial choices may turn out to be bad for later reasoning. Also, handling backward reasoning would be problematic, as initial drawing choices might turn out to be inaccurate. Thus an option to optimise representations for an entire reasoning program would be desirable. This would be an expensive procedure, so only to be applied when requested.

- If we describe object outlines using polynomials, the relational constraints on new objects can be converted into polynomial constraints (e.g. “ $x \in B_r(y)$ ” would become “ $(x-y)^2 - r^2 < 0$ ”). Techniques such as Cylindrical Algebraic Decomposition (CAD) technique could then be used to find ranges of solutions for the new parameters (note that CAD can easily become intractable) [52].
- Solutions would then be tested for the presence of unintentional implicit relations. If such a relation is discovered, its negation can be added as an additional drawing constraint.
- If no solutions exist, constraints would be incrementally removed. When possible, this would be done according to a hierarchy of relations (e.g. it is more important to represent subset relations accurately than order relations). Such a hierarchy should fit with user expectations. This should be investigated empirically, but will probably match the method used for representing a relation (i.e. implicit relations should be represented accurately in preference to explicit graphical relations, which in turn should be preferred to explicit algebraic relations).
- Aesthetic factors could be captured as functions that 'score' a diagram (e.g. objects should have large area, which we could capture by awarding a score based on the average/minimum area of the diagram objects). Given ranges of acceptable parameters, values would then be chosen to optimise these aesthetic factors.

The other option – which would be easier, and possibly interesting in its own right – is to pick *random* parameters when creating universally quantified objects. This would fit in with the 'game semantics' approach to quantifiers, whereby universally quantified objects are picked by an 'enemy' whose choices can be troublesome. Existentially quantified objects would still be drawn by the user.

Note that it is far from clear that fully automated drawing is desirable in general (e.g. it might aid understanding for the user to be in charge of drawing), although it would certainly be interesting to investigate.

8.1.5 Automated reasoning

An automatic theorem proving mode is desirable, both for pragmatic applications (e.g. so that users can skip 'uninteresting' reasoning steps, or in teaching systems for supplying 'proof hints' when students get stuck) and out of academic interest. Automated reasoning in this area is difficult, perhaps largely due to quantifier issues. However there is a fair amount of existing work (e.g. the Theorema system, which utilises a range of techniques [6]). It is not clear at present how hard it would be to adapt such work to our framework. Some applications of automated reasoning also require automated drawing routines (c.f. §8.1.4).

8.2 Extending DDLA

8.2.1 Extending the representation scheme

There are several interesting lines in which the representation scheme could be extended. These include:

- As discussed in §7.2.2, the representation for disjunction could be improved by adding specialised disjunction representations such as spiders.
- A representation for zooming (we envisage that this would be purely 'cosmetic' – i.e. an aid to explaining concepts such as differentiation, but not playing any active role in the reasoning).
- For simplicity, DDLA specifies that object labels are unique meaningless tokens.

However, a more sophisticated approach is both possible and desirable. **Dr.Doodle** already implements the use of meaningful labels for structured objects (e.g. ' $A \cup B$ '; c.f. §6.3.2). More ambitiously, we could also use diagrammatic labels – that is, labels which are mini diagrams. An example of this is shown in Figure 8.1. The graphic objects x' (the red point) and $B_\epsilon(x')$ appear unlabelled. $B_\epsilon(x')$ is then referenced algebraically via a mini-diagram, rather than by label. Such references are clear – provided the mini-diagram has a unique (and obvious) match within the main diagram.

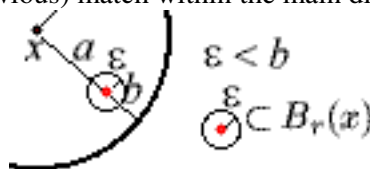


Figure 8.1. Example of a diagrammatic object label (exerted from Figure 4.10).

- More interactivity in representations. (c.f. §5.1.1 which highlights some limitations of DDLA that could potentially be solved with interactive representations). For example, the ability to 'trace through' a diagram, navigating along relations and calling up detailed information on selected objects and relations. More ambitiously, it might be interesting to allow users to vary parameters in rule antecedents, with the system automatically varying the consequent to match. For example, in the “If f is continuous...” rule (c.f. Figure 4.4), the user might vary the size of ϵ , and the system would have to alter δ accordingly. This would give a possibly more appealing demonstration of how the different elements of the rule are linked. This idea is very similar to the variations allowed in Cinderella, although potentially harder to implement due to the presence of existential statements.
- Animated consequents for concepts such as *uniform continuity*:

$$f \text{ uniformly continuous, } \epsilon > 0 \Rightarrow \exists \delta \forall x. f(B_\delta(x)) \subset B_\epsilon(f(x))$$

As stated in §6.2.2, this can already be represented in DDLA by breaking it into two rules. However Figure 8.2 gives an alternative representation using one rule and an animated consequent.

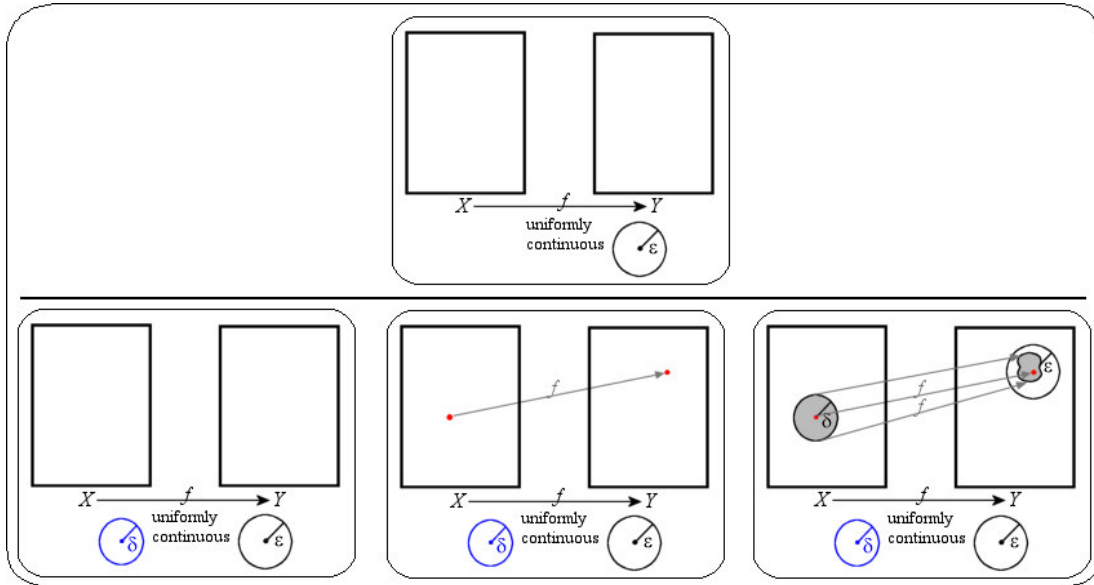


Figure 8.2. Defining uniform continuity using an animated consequent.

- Projection representations for 3D objects (e.g. functions from $\mathbb{R}^2 \rightarrow \mathbb{R}$).
- Representing *error margins*, which we will demonstrate in §8.2.6.

Note that although the representation scheme can potentially be extended to cover many more objects and concepts, there are limits to the extensions possible. Many objects are simply not suited to two dimensional representation (e.g. Klein bottles⁷², \mathbb{R}^n). Hence a complete treatment of analysis is unlikely to be possible without substantial use of algebra.

8.2.2 Using outside reasoning systems

Currently all reasoning in the system must be done using redraw rules. It could improve both ease-of-use and range to allow the incorporation of special 'black box' inference procedures, which can be called for certain classes of sub-problems (e.g. for simplifying arithmetic statements). Such operations would be represented by example transformations. This would be in keeping with the redraw rule representation, but merely illustrative rather than constituting a valid definition.

8.2.3 More accuracy

In counter-example reasoning, we have looked at using diagrams to guarantee the existence of a corresponding model. Currently though this can only be used with diagrams restricted to the $\{\text{type, subtype, } \subset, \in, \text{centre, radius}\}$ relations. Size ($>$ relations) is also suitable for such a treatment. Implicit equational relations and relations of a 'functional nature' (e.g. $a=b$ or $y=f(x)$) are not suitable. This is because of the possibility of exploiting the tiny, but potentially important, differences between objects which are indistinguishable when drawn due to vagueness of representation. However negated versions of such relations *could* be included, since they are not precise (e.g. if a, b are drawn such that $a \neq b$, then this inequality will hold for all objects $a' \approx a, b' \approx b$). To include a relation $r(x,y)$, it is necessary to prove either:

$$\text{implicit}(r(x,y)) \Rightarrow \forall D \in \mathbb{D}^{-1}(\mathcal{D}), D \models r(x,y)$$

(hence vagueness of representation does not matter, as the relation is true for all possibilities. This is the case for “ \subset ” for example, but not “ $=$ ”.)

$$\text{or } \exists \text{ a privileged diagram } D \in \mathbb{D}^{-1}(\mathcal{D}) \text{ such that } \forall \text{implicit}(r'(x,y)), D \models r'(x,y)$$

(this is the case for the centre and radius relations.)

⁷² However whilst a Klein bottle can only be *drawn* in 4D, they *do* have a nice 2D representation which supports a fair amount of reasoning about them. This suggests that with sufficient ingenuity, diagrammatic reasoning might usefully be applied to higher-dimensional objects (i.e. objects that cannot be embedded in \mathbb{R}^3).

A more powerful approach might be to use user-drawn diagrams as a heuristic for generating examples whose accuracy is then checked 'behind-the-scenes'. However, the diagrammatic proof seen by the user would then be only an informal proof, as it would not constitute a sound proof without these behind-the-scenes accuracy checks.

8.2.4 Not rules

Defining a property in DDLA does not define its negation. That is, given a rule defining a property (e.g. 'open'), we do not automatically get a rule defining the negation of that property (e.g. 'not open'). In algebra, moving from a rule to its negation is a simple operation, however for diagrams it is more difficult. Having calculated the algebraic negation, fresh drawings must be created to represent the new rule. Hence automatic negation is dependent on solving the problem of automated drawing. A temporary solution would be to use algebraic representations when the automatic generation of a negation is required.

8.2.5 Sequences

The lack of a mechanism for reasoning about sequences is probably the largest limitation to DDLA's range. In this project we have used a simplistic treatment of sequences as basic objects. This allows us to define convergence, and to prove a couple of general theorems. However there are a great many theorems which cannot be tackled without a way of defining and reasoning about specific sequences. We would like a method that is as visual as possible, and fits in with the representation and reasoning style of DDLA. We propose the following method based on defining sequences by analysing the construction of a few example points.

A sequence can be defined by a redraw rule $R:D \mapsto D'$ where D contains the n^{th} object(s)⁷³, and D' contains the $(n+1)^{\text{th}}$ object(s) in the sequence. Such an R can be used to generate the sequence one point at a time. It can also be used to reason inductively about properties of the sequence.

Such sequence-generation rules can be created by example. If a user constructs the first few points of the sequence, the reasoning program produced by this will then contain example

⁷³ Often several objects will be created at each step. This is because a sequence of points may also require sequences of 'supporting objects' used in the construction.

generation steps which can be used to define the rule R . R must then be checked to make sure that it does define a sequence (i.e. that it can be applied indefinitely to generate an infinite chain of points). This check is straightforward: if $D' \simeq D$ under the matching given by $(n+1) \rightarrow n$, then R can be replied indefinitely. This definition method also covers 'conditional sequences' (i.e. where there is a case split in creating the $(n+1)^{\text{th}}$ point), for example the sequences used in repeated bisection, and sequences of sets and functions as well as points.

We can then reason about properties of such sequences in at least two ways.

- 1) Transitive properties such as $x_{n+1} > x_n$ can be extended throughout the sequence: e.g. if $x_{n+1} > x_n$, then $m > n \Rightarrow x_m > x_n$.
- 2) By induction, we can conclude that a relation which is true for x_N (any N) and preserved by R will be true for all x_n , $n > N$. This requires making proof-by-induction available. This could be done either as an animated rule or as a meta-rule (which would allow us to develop a specialised representation – probably desirable for a concept as important and confusing as induction).

In Appendix B, we illustrate this method with an example proof.

8.2.6 Differentiation

For functions from $\mathbb{R} \rightarrow \mathbb{R}$, there is an obvious graphical representation for the derivative as the gradient of the function's graph. Defining the property of being differentiable is trickier though. The standard definition is:

$$f \text{ is differentiable at } x \text{ with derivative } f' \text{ if} \\ f(x+h) = f(x) + h.f' + e(h)|h|, \text{ where } e(h) \rightarrow 0 \text{ as } h \rightarrow 0$$

This is equivalent to:

$$f \text{ is differentiable at } x \text{ with derivative } f' \text{ if } \forall \epsilon > 0 \\ \exists \delta > 0 \text{ such that } h < \delta \Rightarrow |f(x+h) - f(x) - h.f'| / h < \epsilon$$

This is now in a form which we can capture using the current framework, since it is equivalent to saying that the function $g(y) = |f(y) - f(x) - (y-x).f'| / (y-x)$ is continuous at x . However this is not an intuitive definition. A more appealing diagrammatic definition is possible (Figure 8.4) but requires an extension to our representation scheme, which we now present.

Error margins

We can think of $f(x) + h.f'$ as an *approximation function* for $f(x+h)$, and add the concept of *acceptable error*, which we represent by shading a region of the graph around the approximation function. This shaded region is the error margin. Typically, an approximation will have a range within which it is valid, and this determines the width of the shaded region. Plotting the true function within the shaded region represents (implicitly) that the approximation is within acceptable error. Figure 8.3 shows an example of an error margin, with acceptable error ϵ (in this case, the approximation is not valid, as the true function breaks the error margin).

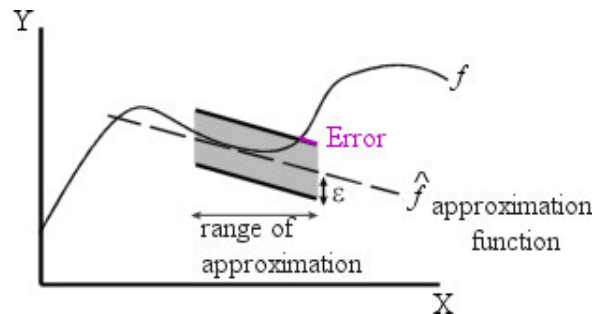


Figure 8.3. An approximation function with a constant error margin.

For differentiability, the acceptable error between $f(x+h)$ and $f(x)+h.f'$ is $h.\epsilon$. This allows us to define differentiability as shown in Figure 8.4.

We can also define the property of continuity for 1D functions in this way, as shown in Figure 8.5. This then gives a visual demonstration of the difference between continuity and differentiability, from which it is clear that f differentiable $\Rightarrow f$ continuous (indeed, contrasting the diagrammatic definitions is almost a proof).

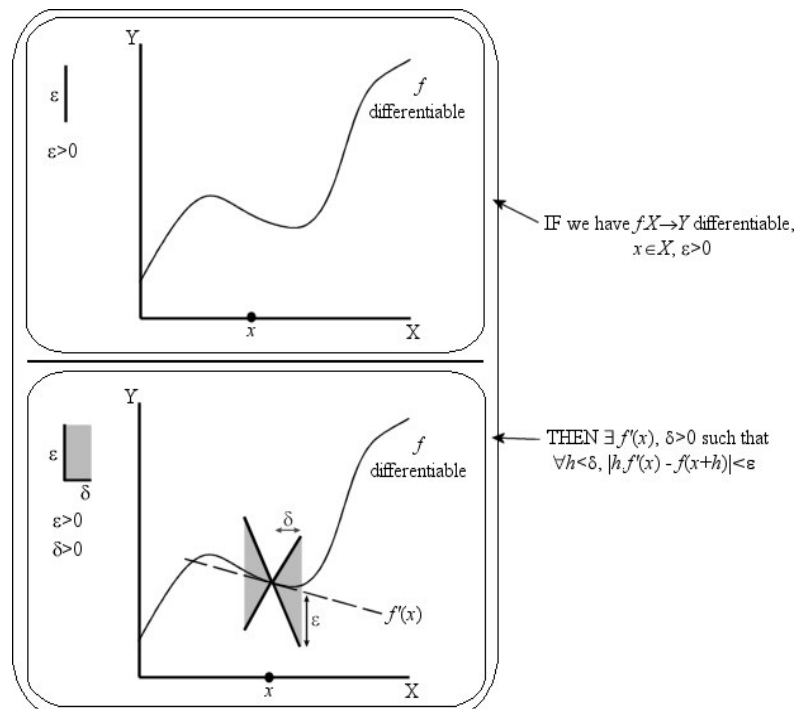


Figure 8.4. Defining differentiability using error margins.

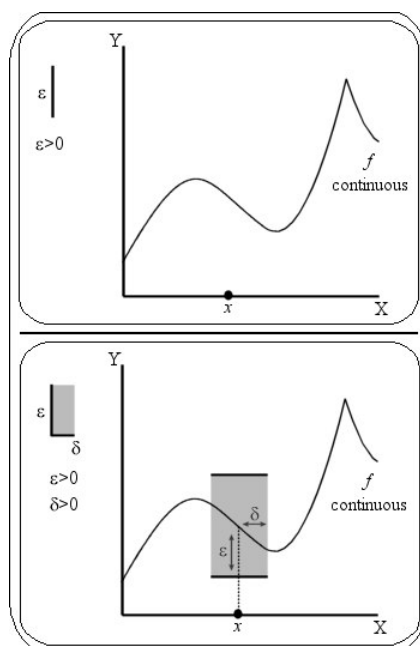


Figure 8.5. Defining continuity using error margins.

Note that when working algebraically on questions of differentiability, one often shifts the origin such that $x=f(x)=0$ to make the statements simpler. With a diagrammatic

representation, people naturally shift their focus to $(x, f(x))$. We could say either that an origin shift is unnecessary, or that it is performed implicitly.

8.2.7 Integration

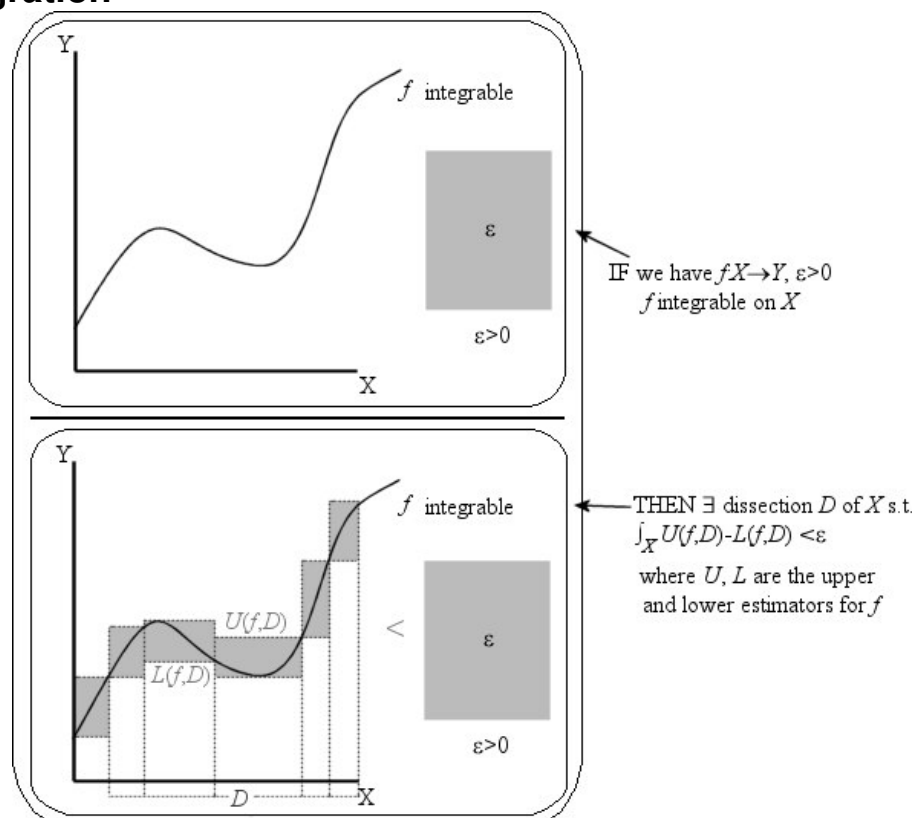


Figure 8.6. Diagrammatic definition for integrability.

As with differentiation, there is an obvious graphical representation for the integral of a function from $\mathbb{R} \rightarrow \mathbb{R}$ (as the area under the function's graph), but it is less clear how to define the property of being integrable. If we stick to the usual definition (the Riemann integral, using upper and lower sums), then one possible representation would be based on comparing the total area between the upper and lower sums to an arbitrary ϵ , as shown in Figure 8.6. This ties in nicely with the error-margin representation suggested for differentiability and continuity in §8.2.6. A simple and intuitive proof is then possible for the important theorem “ $f: \mathbb{R} \rightarrow \mathbb{R}$ continuous, $X \subset \mathbb{R}$ bounded $\Rightarrow f$ integrable over X ” (see Figure 8.7).

8.3 Summary

This chapter has set out several ways in which this project can be extended.

One straightforward line of work is to extend and improve the **Dr.Doodle** system. We have identified various possible improvements to usability, including automated drawing, automated reasoning and interface improvements. Future work here also includes investigating how the different features of the system are used in practice, and what effect different settings/configurations have on users.

Perhaps more important, though, is the question of how to extend DDLA. We have proposed several ways of increasing the range and power of the logic. These include a mechanism for defining sequences, and a new representational device – error margins – that seems to give visually appealing representations for several key concepts. Developing these ideas further should allow us to produce intuitive proofs for an ever wider range of theorems.

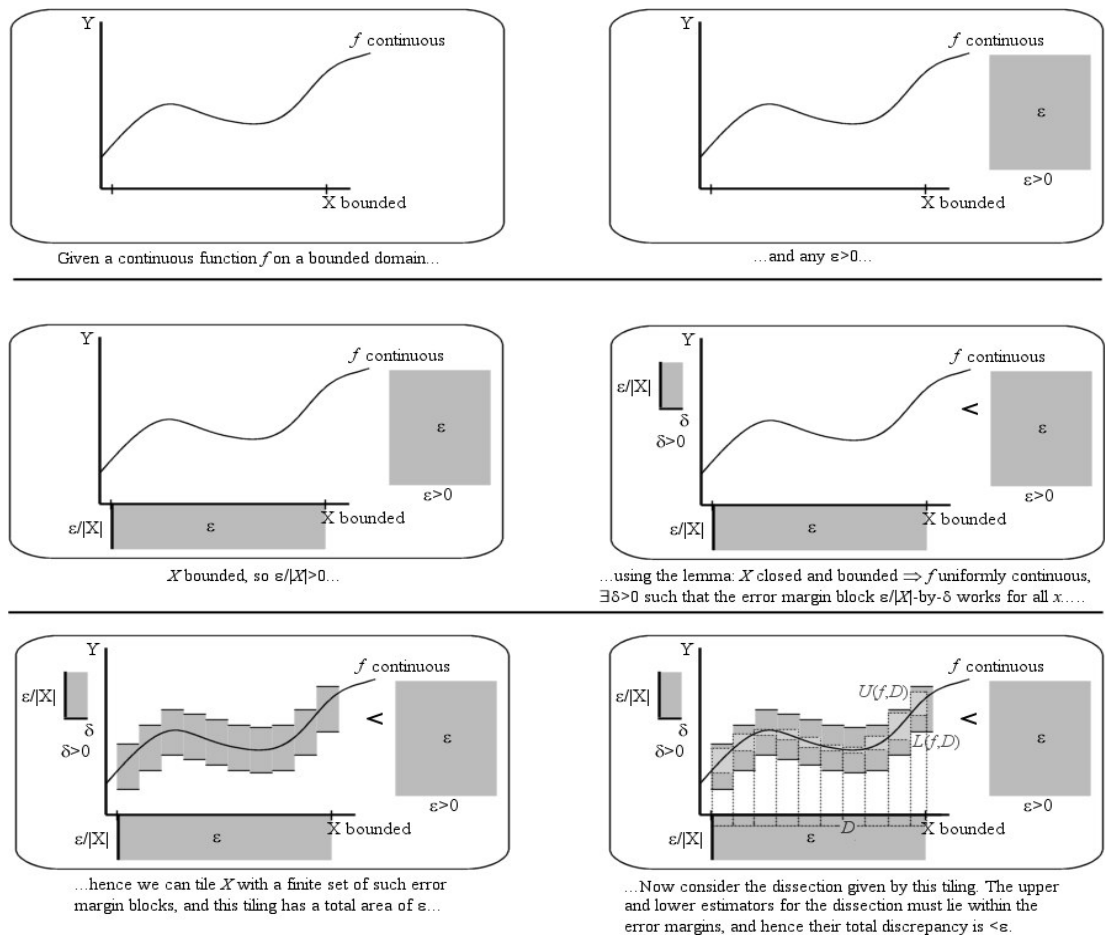


Figure 8.7. Diagrammatic proof that f continuous, X closed & bounded $\Rightarrow f$ integrable on X .

9 Conclusion

This project has looked at using diagrammatic reasoning to prove mathematical theorems. Behind this work lie two beliefs: that diagrammatic reasoning is more intuitive for some domains, and that if theorem-provers are to be more than black boxes, then we must make computers reason like people. The alternative – that people should reason like computers – is unrealistic, however attractive it may seem in theory. Of necessity, we have focused on a specific domain (mathematical analysis). This is perhaps a more challenging domain than has been tackled in diagrammatic reasoning to date. The aims of the project were to develop a way of reasoning with diagrams suitable for doing analysis proofs, and demonstrate that this has advantages over conventional methods. Whilst this work is not yet complete, we have made considerable progress, showing that these aims are both possible and desirable.

9.1 Aims

Since Hilbert's pioneering work, axiomatic algebraic approaches have taken over mathematics. Whilst this revolution has delivered many benefits, it has also led to dry formalisations that lack intuitive appeal. Algebraic definitions can be opaque and meaningless when first met, and even quite simple reasoning steps can become difficult to follow when couched in dry symbolic terms. The effect is to make geometric subjects much more daunting to newcomers than is necessary.

This problem is even worse in computer systems. Modern theorem provers are often hard to use and their proofs unreadable, even for experts. We believe this severely limits their usage in mathematics. One aspect which they generally lack is the ability to support visualisation of ideas and concepts. Yet it is with computers that the future of diagrammatic reasoning lies, and computers could revolutionise visual reasoning. Drawing programs can make it possible for everyone to create visualisations quickly and accurately. Also, using computers opens up exciting new possibilities for diagrammatic reasoning - such as animated diagrams and interactive diagrams.

However, as we explored in chapter 3, there are serious obstacles involved in formalising diagrammatic reasoning. These include a certain unavoidable roughness in the representation, optical illusions, and ambiguous drawings. Also, several straightforward logical ideas are hard to represent diagrammatically (e.g. disjunction, negation, proof by contradiction and handling quantifiers), and generalisation is problematic (which is linked to specifying exactly what a diagram does and does not represent). Plus, we must be careful to keep design issues in mind. Otherwise it is possible to create logics that are actually less intuitive than algebraic approaches. This range of issues makes formalising diagrammatic reasoning an interesting challenge.

We focused on a domain that has not been tackled before, that of Euclidean-space analysis. Euclidean plane geometry has always been taught using diagrammatic reasoning. This project has looked at using diagrams to prove properties in a more abstract geometry. We have tackled a subset of Euclidean space analysis. This is algebraically a hard domain, and even great mathematicians such as Cauchy have made mistakes in this subject.

9.2 Our work

9.2.1 Exploration

The first stage of the project was exploratory. Unlike the DIAMOND project, an established body of informal representations and proofs was not available. By tackling a range of analysis theorems, we have built up a body of new diagrammatic representations and proofs. These then guided our formalisation, and provided content for the final system.

9.2.2 Formalisation

The framework we developed is:

- Wholly diagrammatic – rules and proofs are both presented diagrammatically.
- ...but heterogeneous – diagrams mix visual and sentential elements.
- and *dynamic* – both reasoning and representation involve the drawing *process*, rather than interpreting a finished diagram.
- Its inference mechanisms work at a syntactic level.
- ...but can leverage semantic information from the representations. This is used to simplify counter-example proofs, and to formally implement a version of Lakatos's

method of *strategic withdrawal* whereby a conjecture can be weakened as necessary whilst trying to prove it.

Using rules for *implicit relations*, we can specify which facts can be inferred from the diagram, and *implicit inferences* allow some proof steps to be skipped. Combining this with *explicit relations* that can act as 'algebraic overrides', gives us the flexibility to alter implicit relations where necessary. This provides a way of formalising reasoning from the diagram which allows for intuitive understanding with reasonable range. The wholly diagrammatic nature of the reasoning – with even the rules defined by diagrams – is rare.

Within this framework, we then built a representation scheme for analysis concepts. We have presented designs for objects and relations in the domain, with some discussion of their strengths and weaknesses. The design of these representations was guided by the aim of combining intuition, representational range and reasoning power. However in places, conflicts between these different design goals have led to compromises.

This framework was then analysed for soundness. This involved specifying a way of converting diagrams into algebra, so that our diagrammatic rules and proofs can be compared with the standard definitions of the domain. We have shown that our logic is sound - subject to a meta-assumption of reasonable clarity in the diagrams. This assumption is unavoidable in most diagrammatic reasoning. However we can hope that by analysing the drawing and reading of diagrams, we may develop theories allowing us to specify when a diagram is clear.

9.2.3 Implementation

An important part of this project has been the building of an interactive theorem prover ('Dr.Doodle') to implement this logic. This served several goals. Firstly, the development of DDLA and Dr.Doodle went hand-in-hand to a certain extent, with work on constructing Dr.Doodle helping to guide the formalisation of DDLA. The finished system serves as a proof of concept. It demonstrates that our logic works, and does not contain any 'magic steps', which could easily remain hidden in a paper theory. Moreover, it is also a useful tool for testing claims regarding the pedagogic value of diagrammatic reasoning in this domain.

It has allowed us to perform experiments and hence an empirical evaluation. Finally, it serves as a base upon which a real-world teaching aid could be built.

9.3 Assessment

We evaluated this project against the hypothesis:

“Diagrammatic proofs are possible for analysis problems, and may be easier (in some sense) than algebraic reasoning”

Our conclusions supported this hypothesis. The experiments using **Dr.Doodle** to teach undergraduates were particularly promising. Students using diagrammatic reasoning consistently out-performed those using algebra. This shows that the ideas presented here have genuine pragmatic value.

Our evaluation was limited in scope however. We only examined proof *production*. Communication and recall, both of proofs and concepts, are also of interest. Nor have we examined *why* diagrammatic reasoning is better, or which aspects of our logic are responsible for the improvement in performance. Ultimately we would like to have an empirically validated theory for when diagrams are better than algebra – and what sort of diagrams are best.

Comparing our work here with related research has shown strong links, but also large differences. Whilst ideas from one area can be fruitfully applied elsewhere, none of the systems we looked at could be considered 'rivals'. Such comparisons suggest that diagrammatic systems are inherently specialised, either for a domain or a type of problem. We should therefore expect a plethora of computerised diagrammatic reasoning systems to emerge, rather than for development to converge on one reasoning system. There are theories that attempt to categorise the different types of diagram and diagram logic, often based on cognitive ideas. Arguably though, it is too early to do so: we must first see what becomes of current projects, before we can say what aspects of a diagram logic work and why.

9.4 Where next?

9.4.1 Extensions to DDLA

In §8 we looked at how this work could be extended. This section showed that there is the potential to do much more with diagrammatic reasoning in this domain than has been done here. This project constitutes a promising start rather than a finished body of work. However I see no reason why our hybrid diagrammatic-algebraic logic cannot be extended to cover the whole of analysis.

9.4.2 Other applications

Any project, however focused, both draws ideas from other fields, and offers ideas back in return. We have used animation to give meaningful quantifier representations. I suspect that other applications of animation are possible: although unrelated to our present work, animated diagrams may also be useful in representing and reasoning about temporal relations. There is an obvious attraction in using time to represent itself.⁷⁴

Ideas from this project could also be applied in other areas. One possible line of work is in interactive program generation. As the term 'reasoning-program' suggests, a proof in our logic is in fact a program for transforming diagrams (complete with conditions on when it is valid), and animated pre-conditions can be thought of as program specifications (with flexible links being unwritten bits). If we work with program objects (e.g. lists, strings, class instances) instead of graphic objects, this becomes a programming method. Hence our work suggests a new approach to the important-but-difficult problem of programming-by-example. We explore this idea further in [59].

9.5 Closing thoughts

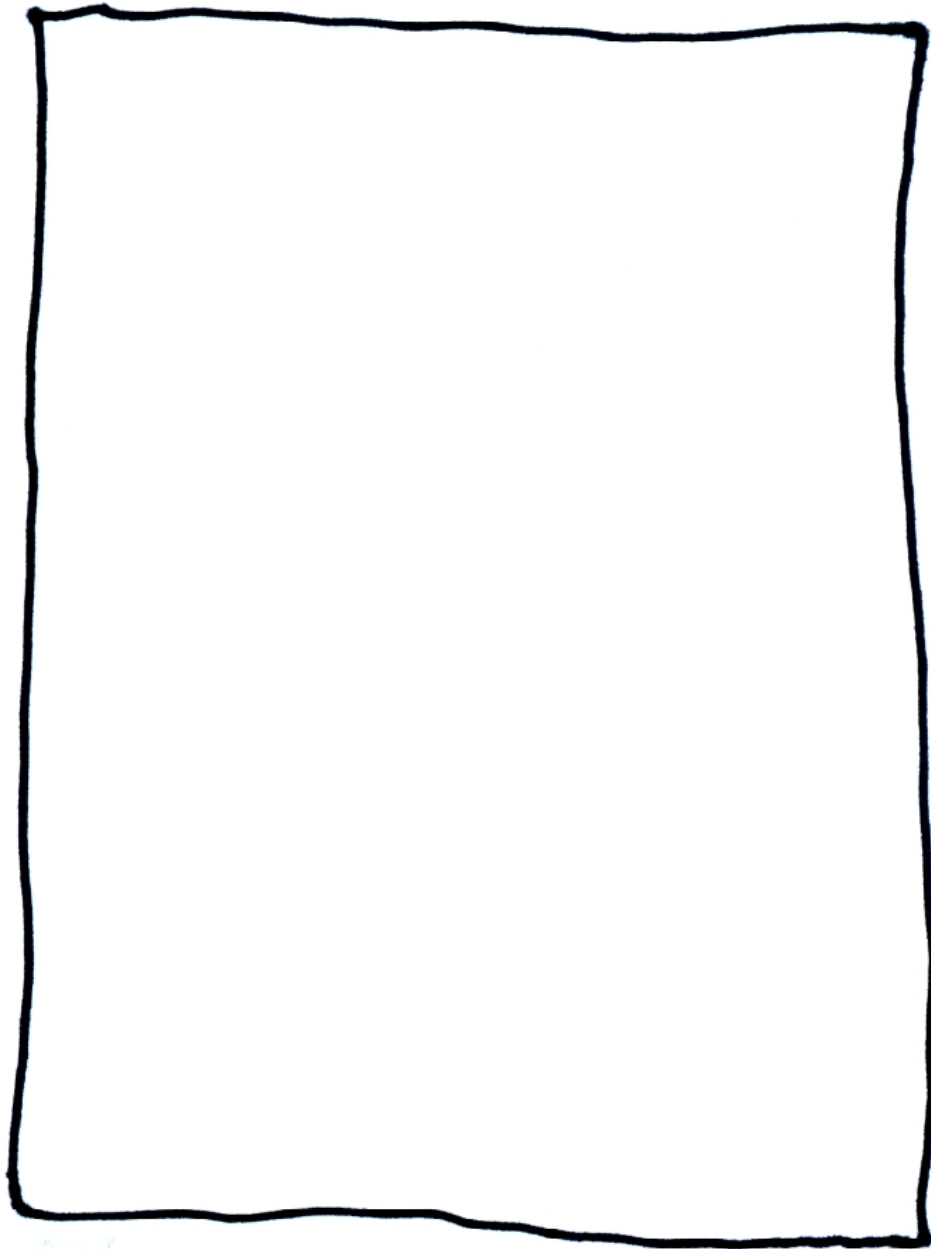
Although once neglected, the field of diagrammatic reasoning is now attracting serious attention. Potentially, it could dramatically improve the usability of many computer systems, including theorem provers and programming environments. There is a lot of work to be done though, and a lot of basic questions that are still open. Euclidean plane geometry has always been taught using diagrammatic reasoning. This project has shown that diagrammatic

⁷⁴ This would probably not be suitable for domains which involve precise time calculations, as these would be hard to judge in an animation. For qualitative reasoning though, or as part of a mixed system, it seems an interesting line for future research.

reasoning can also be used in a more abstract geometry. Hopefully, this work will serve as a foundation for a visual formalisation of analysis.

By drawing on diagrams to visualise concepts, we hope that more intuitive formalisations of geometry are possible than at present: logics that give rigorous proofs, yet are nevertheless suitable for teaching with and supporting thinking. Eventually we envisage that formal proofs, produced or verified by computers, will be as readable as human proofs. At that point, theorem provers will truly become 'assistant mathematicians'.

This space left blank
for you to draw your
own conclusions



10 Glossary of Terms

This glossary is provided as a useful reference for the technical sections of chapters 4 and 5. The definitions given here are short *aide-memoires*. They neither reproduce nor replace those of chapters 4 and 5, or make sense without the explanations given in those chapters.

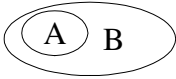
10.1 Notational conventions


- Upper case letters (sometimes with subscripts or primes) are used for diagrams, redraw rules and (within the context of a specific diagram) sets.
- Underlined upper case letters are used for reasoning programs.
- In the context of reasoning programs, let $D - D'$ denote that D' is a child node of D .
- Lower case letters (sometimes with subscripts or primes) are used for graphic objects.
- Relations are written in the form $r(x_1, x_2, \dots, x_n)$ where r is the relation name and x_i are object names or constants. Relations may have any arity. However, since binary relations are the most common, we will allow $r(x, y)$ to represent an arbitrary relation.
- \Rightarrow has its standard logical meaning of “implies”.
- \simeq means “matches” e.g. “ $A \simeq B$ ” is used for A matches B
- $\simeq_m \simeq$ means “matches with mapping m ”. Mapping functions are overloaded so that the same name designates several different functions depending on the type of input.
- \mapsto means “redraws to” e.g. “ $D \mapsto D'$ ” is used for “ D redraws to D' ”
- $\overset{R}{\mapsto}$ means “redraws using rule R ”

10.2 Terms defined in this project

The following list covers the terms we have defined in this project. Terms are listed in alphabetical order. The *example(s)* column shows typical statements that might be made involving these terms. The *defn.* column gives page number references to the sections(s) where the term is defined. Where multiple references are given, early references will typically be informal definitions.

Term	Example(s)	Quick definition	Defn.
Animated rule	$R = D_0 \dots D_n \mapsto D'$	An animated rule has a chain of diagrams as its pre-condition. The diagram chains are used to represent and reason about quantifiers. The chains use two different types of link, <i>strict</i> and <i>flexible</i> , corresponding to universal and existential quantification.	p. 74, 91
Branch rule	$R = D_0 \mapsto D_1 \dots D_n$	Redraw rule that introduces a case split by branching the reasoning program.	p. 71, 91
Diagram	$D = (\text{objects}, \text{relations}), D_1 \dots D_n$	A set of <i>graphic objects</i> and a bag of <i>observed relations</i> .	p. 74, 91
Domain		Euclidean space analysis.	
Drawing function \mathbb{D}	$\mathbb{D}(D)$	A function mapping idealised objects onto bitmaps, and by extension, creating physical diagrams from idealised diagrams.	p. 126, 131
Explicit relation	$\text{open}(X)$	A relation that requires the diagram objects to be annotated (graphically or algebraically) in order to be represented.	p. 85
Flexible link	$D_1 \text{ -f- } D_2$	Used in animated rules to indicate that the transition introduces \exists objects.	p. 74, 91
Graphic Object		A labelled domain object.	p. 84
Idealised diagram	D	A diagram of geometric objects (as opposed to <i>physical objects</i>). Idealised diagrams are usually synonymous with diagrams.	p. 126

Term	Example(s)	Quick definition	Defn.
Idealised object	$x \in \mathbb{R}, x = 4.32156$	A geometric object (as opposed to the drawing of a geometric object, which is a <i>physical object</i>). Idealised objects are synonymous with graphic objects.	p. 126
Interpretation function \mathbb{I}	$\mathbb{I}(R) = "\forall x.p(x) \Rightarrow q(x)"$	A function mapping from (idealised) diagrams to algebra.	p. 132
Implicit inference rule	$A \subset B, B \subset C \Rightarrow A \subset C$	Rule capturing a 'free ride'; an inference that happens simply by drawing the diagram objects.	p. 86
Implicit relation	 $(A \subset B)$	A relation that is represented simply by drawing the diagram objects.	p. 85
Label	x, A, f	A token associated with a graphic object. Labels must be unique within the surrounding reasoning program.	p. 84
Matching	$T \simeq_m \simeq D$	A <i>matching</i> is a mapping from the objects of one diagram to another, such that the relations of the source diagram also hold in the target diagram. Matchings can be many-to-one, and are not generally symmetric.	p. 88,94
Model	\mathbb{M}	Two uses: 1) An (assumed) model for euclidean space analysis which includes \mathbb{R}^2 and all the drawable objects. 2) A subset of this model containing the objects used in an idealised diagram.	p. 126, 130

Term	Example(s)	Quick definition	Defn.
Observed relation	$r(x,y) \in \text{relations}(D)$	A relation that is known to be true in this diagram. Observed relations can be either <i>implicit</i> or <i>explicit</i> .	p. 85
Physical diagram	$\mathcal{D} = \mathbb{D}(D)$	A diagram of physical objects, created from an idealised diagram.	p. 126
Physical objects		The physical drawing of a geometric object.	p. 126
Redraw rule	$R = D \mapsto D'$	A visual equivalent to rewrite rules. A redraw rule has a diagrammatic antecedent (possibly animated) and consequent (possibly branching). These define an example drawing operation. Applying the rule to a matching diagram performs an equivalent drawing operation on the target diagram.	p. 69, 89
Simple rule	$R = D \mapsto D'$	A non-animated non-branching redraw rule.	p. 90
Specific object	$\text{specific}(X)$	A specific object is both existentially quantified and fully observed, which means that any relation which can be seen to be true for that object, can be used in the reasoning. This allows some shortcuts in a counter-example proof.	p. 98
Strict link	$D_1 \text{ -s- } D_2$	Used in animated rules to indicate that the transition is exact, introducing \forall objects.	p. 74, 91
Unknown	$\text{unknown}(A \subset B)$	Annotation specifying that a relation appearing in the diagram is accidental, and should be ignored.	p. 86

11 References

- [1] D.Barker-Plummer & S.C.Bailin “On the Practical Semantics of Mathematical Diagrams” in *Reasoning with Diagrammatic Representations II*, AAAI, 1997.
- [2] D.Barker-Plummer & S.C.Bailin, & Ehrlichman “Diagrams and Mathematics” draft copy of an unpublished paper, Stanford University, 1995.
- [3] Barwise & Etchemendy “Heterogeneous Logic” in *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, AAAI Press/The MIT Press, 1995.
- [4] A.Bierce “Fantastic Fables: The Flying Machine” 1899. Project Gutenberg e-text 1995, <ftp://ftp.ibiblio.org/pub/docs/books/gutenberg/etext95/fanfb10.txt>
- [5] A.Bogomolny “Plane Filling Curves” from website <http://cut-the-knot.com/>, 1999.
- [6] B.Buchberger, C. Dupre, T. Jebelea, F. Kriftner, K. Nakagawa, D. Vasaru & W. Windsteiger “The Theorema Project: A Progress Report” in *Symbolic Computation and Automated Reasoning*, A.K.Peters Ltd., 2000.
- [7] B.Chandrasekaran “What Does It Mean for a Computer to Do Diagrammatic Reasoning?” Keynote talk to Diagrams 2002, published online at <http://www.cis.ohio-state.edu/~chandra/>, 2002.
- [8] T.Eisenberg & T.Dreyfus “On the Reluctance to Visualize in Mathematics” unpublished paper available from Weizmann Institute of Science, quoted in [46], 1990.
- [9] A.Fish & J.Howse “Towards a Default Reading for Constraint Diagrams” proceedings of the 3rd International Conference on Theory and Application of Diagrams, Springer-Verlag, 2004.
- [8] J.D.Fleuriot & L.C.Paulson “A Combination of Nonstandard Analysis and Geometry Theorem Proving, with Application to Newton's Principia” 1998.
- [9] B.Fraser “Structuralism” course notes, Cambridge University. Available online at <http://www.classics.cam.ac.uk/Faculty/structuralism.html>, 1999.
- [10] C.Gurr, J.Lee & K.Stenning “Theories of Diagrammatic Reasoning: Distinguishing Component Problems” in *Minds & Machines* 8(4), Kluwer Academic, 1998.
- [11] J.Gil, J.Howse & S.Kent “Towards a Formalization of Constraint Diagrams” IEEE, 2001.

- [12] J.Gil, J.Howse, S.Kent & J.Taylor “Projections in Venn-Euler Diagrams” in proceedings of “Visual Languages”, 2000.
- [13] M.Greaves “The Philosophical Status of Diagrams” CLSI publications, Stanford, 2002.
- [14] E.Hammer “Reasoning with Sentences and Diagrams” in *Notre Dame Journal of Formal Logic*, 1994.
- [15] E.Hammer “Logic and Visual Information” CSLI publications, Stanford, 1995.
- [16] P.J.Hayes. Introduction to “Diagrammatic Reasoning: Theoretical Foundations” in *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, AAAI Press/The MIT Press, 1995.
- [17] P.J.Hayes & G.L.Laforte “Diagrammatic Reasoning: Analysis of an Example” in *Formalizing Reasoning with Visual and Diagrammatic Representations*, AAAI Press, 1998.
- [18] J.Heiser & B.Tversky “Descriptions and Depictions in Acquiring Complex Systems” proceedings of the Annual Meeting of the Cognitive Science Society, 2002.
- [19] C.G.Hempel “Geometry and Empirical Science” *American Mathematical Monthly* 52, 1945.
- [20] C.G.Hempel “On the Nature of Mathematical Truth” *American Mathematical Monthly* 52, 1945.
- [21] R.E.Horn “Visual Languages” MacroVU Press, 1998.
- [22] J.Howse, F.Molina, J.Taylor, S.Kent & J.Gil. “Spider diagrams: A diagrammatic reasoning system” in *Journal of Visual Languages and Computing*, 2001.
- [23] D.Icove, K.Seger & W.VonStorch “Computer Crime, A Crimefighter's Handbook”, O'Reilly Publishing, 1995.
- [24] M.Jamnik “Automating Diagrammatic Proofs of Arithmetic Arguments” PhD thesis, Edinburgh Division of Informatics, 1998.
- [25] M.Jamnik, A.Bundy & I.Green “On Automating Diagrammatic Proofs of Arithmetic Arguments” *Journal of logic, language and information*, 1999.
- [26] G.Klima “Existence and Reference in Medieval Logic” in *New Essays in Free Logic* editors A.Hieke & E.Morscher, Kluwer Academic, 2001.
- [27] T.W.Korner “Analysis” lecture notes, Cambridge University mathematics tripos part Ib. available online at <http://ftp.dpmms.cam.ac.uk/pub/twk/Anal.ps>, 1998.
- [28] I.Lakatos “Proofs and Refutations” Cambridge University Press, 1976.
- [29] H.Lieberman, “Mondrian: A Teachable Graphical Editor” in *Watch What I Do: Programming by Demonstration*, editor Allen Cypher, MIT Press, 1993.

- [30] E.Maclean, J.Fleuriot & A.Smail "Proof-planning Non-standard Analysis" proceedings of the Seventh International Symposium on Artificial Intelligence and Mathematics, 2002.
- [31] K.Marriott "Formal Approaches to Visual Language Specification and Understanding", invited speaker at the 1st International Conference on Theory and Application of Diagrams, 2000.
- [32] E.A.Maxwell "Fallacies in Mathematics" Cambridge University Press, 1959.
- [33] R.E.Mayer & J.K.Gallini "When is an Illustration Worth Ten Thousand Words?" Journal of Educational Psychology, Vol. 82 (4), pp715-726, 1990.
- [34] L.Meikle & J.Fleuriot "Formalizing Hilbert's Grundlagen in Isabelle/Isar", proceedings of the 16th conference on Theorem Proving in Higher Order Logics (TPHOLs), Springer-Verlag, 2003.
- [35] B.Meyer. "Constraint Diagram Reasoning" unpublished paper, available online at the authors website, 2000.
- [36] B.Meyer, H.Zweckstetter, L.Mandel & Z.Gassmann "Automatic Construction of Intelligent Diagrammatic Environments" proceedings of 8th International Conference on Human-Computer Interaction, 1999.
- [37] R.B.Nelsen "Proofs Without Words" The Mathematical Association of America, 1993.
- [38] J.Oberlander, R.Cox & K.Stenning "Proof styles in multimodal reasoning" in *Logic, Language and Computation*, editors J.Seligman & D.Westerståhl, Stanford CSLI Publications, 1996.
- [39] J.J.O'Connor & E.F.Robertson. "The MacTutor History of Mathematics Archive" <http://www-history.mcs.st-and.ac.uk>, 1999.
- [40] L.Penrose & R.Penrose "Impossible objects: A Special Type of Visual Illusion" British Journal of Psychology, vol. 49, 1958.
- [41] J. Richter-Gebert & U.Kortenkamp "The Interactive Geometry Software Cinderella" Springer-Verlag, 1999.
- [42] N.Rose "Mathematical Maxims and Minims" Rome Press Inc., Raleigh, NC, USA, 1988.
- [43] M.I.Sereno "Language & the Primate Brain" proceedings, 13th Annual Conference of the Cognitive Science Society, pp79-84, Lawrence Erlbaum Association, 1991.
- [44] L.K.Schubert "Extending the Expressive Power of Semantic Networks" in *Artificial Intelligence* 7(2) pp163-198, Elsevier, 1976.
- [45] A.Shimojima "Operational Constraints in Diagrammatic Reasoning" in *Logical Reasoning with Diagrams*, editors Barwise & Allwein, OUP, 1996.
- [46] S.J.Shin "The Logical Status of Diagrams" Cambridge University Press, 1995.

- [47] S.Singh “Fermat's Enigma: The Epic Quest to Solve the World's Greatest Mathematical Problem” Bantam Books, 1998.
- [48] H.Simon & J.Larkin “A Diagram is (Sometimes) Worth 10,000 Words” Journal of Cognitive Science, 1987.
- [49] A.Sloman “Interactions Between Philosophy and A.I.: The Role of Intuition and Non-Logical Reasoning in Intelligence” proceedings 2nd International Joint Conference on Artificial Intelligence, 1971.
- [50] A.Sloman “Diagrams in the Mind?” proceedings of Thinking with Diagrams, published in *Diagrammatic Representation and Reasoning*, editors M.Anderson, B.Meyer & P.Olivier, Springer-Verlag, 1998.
- [51] K.Stenning & P.Monaghan “Effects of representational modality and thinking style on learning to solve reasoning problems” in Proceedings of the 20th Annual Cognitive Science Society Conference pp716-721.
- [52] A.Strzebonski “Cylindrical Algebraic Decomposition” in *E.Wiesstein's World of Mathematics*, <http://mathworld.wolfram.com/CylindricalAlgebraicDecomposition.html>, 2003.
- [53] D.Tall. “The Cognitive Development of Proof: Is Mathematical Proof For All or For Some?” presented at the Conference of the University of Chicago School Mathematics Project, 1998.
- [54] H.Traunmuller “Conversational Maxims and Principles of Language Planning” unpublished paper, available online at <http://www.ling.su.se/staff/hartmut/griceil.htm>, Department of Linguistics, Stockholm University, 1991.
- [55] D.Wang “Geometry Machines: From AI to SMC” proceedings of the 3rd International Conference on Artificial Intelligence and Symbolic Mathematical Computation (AISM3), Springer-Verlag, 1996.
- [56] R.White “Learning Theory and the Science Curriculum” proceedings of the *Science Years 7–10 Symposium, Board of Studies, NSW*, 1996.
- [57] D.Winterstein, A.Bundy & M.Jamnik “On Differences Between the Real and Physical Plane” proceedings of the 3rd International Conference on Theory and Application of Diagrams, Springer-Verlag, 2004.
- [58] D.Winterstein “On Differences Between the Real and Physical Plane: Analysis of the inside relation” Informatics Report Series, Edinburgh University, 2003.
- [59] D.Winterstein “A Proposal for Interactive Program Generation” Informatics Report Series, Edinburgh University, 2003.
- [60] Wen-tsun Wu. “Basic Principles of Mechanical Theorem Proving in Elementary Geometries”, 1985.

- [61] Chuan-Zhong Li & Jing-Zhong Zhang “Readable Machine Solving in Geometry & ICAI Software MSG”
- [62] “Diagrammatic Reasoning: Cognitive and Computational Perspectives” editors J.Glasgow, N.Hari Narayanan, and B.Chandrasekaran AAAI Press / MIT Press, 1995.
- [63] “The World of Mathematics” editor J.R.Newman, Simon&Schuster, 1956.
- [64] “Diagrammatic Representation and Inference” editors M.Hegarty, B.Meyer, N.Hari Narayanan Springer-Verlag, 2002.
- [65] “Fundamentals of Analysis” course exams, Edinburgh University Mathematics Department, 2001-2003.
- [66] Undergraduate syllabi, Edinburgh University Mathematics Department, available online at <http://www.maths.ed.ac.uk/guides>, 2003.

Appendix A: Rule-set for DDLA

12.1 Appendix overview

This appendix presents the rule-set used in DDLA/Dr.Doodle, including the implicit inference rules. Rules are grouped roughly according to the objects and relations involved. This appendix also gives soundness proofs for these rules, based on the framework of chapter 5. In most cases, the rules are visual interpretations of conventional definitions, and hence soundness follows trivially from Theorem 17 (c.f. §5.2.2). For such rules, we simply give the algebraic interpretation of the rule.

The redraw rules are mostly presented using screen-shots taken from Dr.Doodle (re-arranged slightly to fit the layout used in the thesis). Therefore, as described in §6.3.2, they show some minor discrepancies from the DDLA specification set out in chapter 4. In a few rules, the graphical elements play very little role. In these cases, the rule is presented algebraically. Implicit inference rules are also presented algebraically.

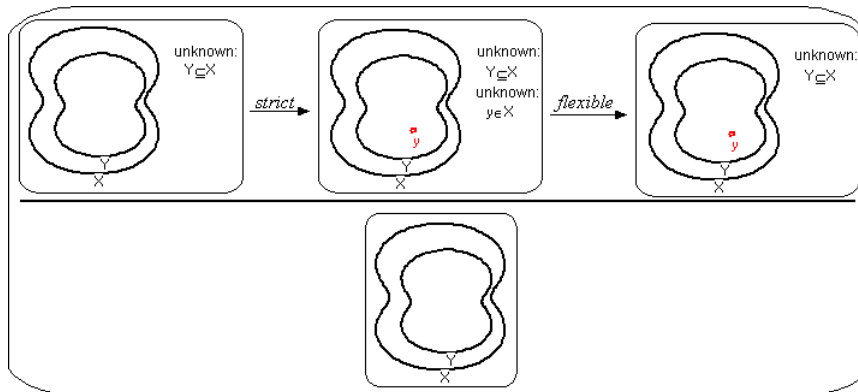
This rule-set is not complete. It was extended as necessary, and there are rules which are missing because they were not used in any of our proofs. Where other rules are used in our proofs, these are either:

- 1) Lemmas, provable with this rule-set.
- 2) Rules creating universally quantified objects. We do not list them here because the necessary rules are embedded in the animated rules, and can be automatically (and very easily) extracted from these (c.f. Definition 4.3.3.3, §4.3.3).

12.2 Rule-set with soundness proofs

12.2.1 Objects

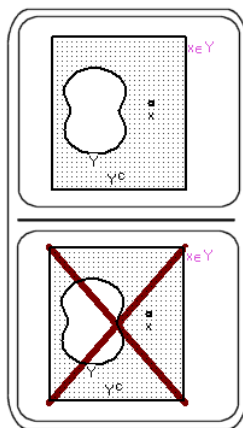
Sets



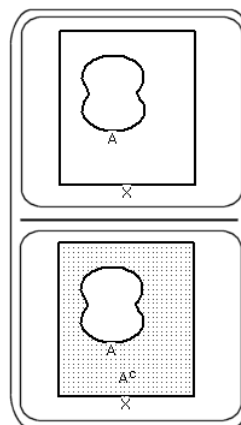
Rule 1. Subset - recognise.

Rule 2. Subset – apply (an implicit inference rule): “ $X \subset Y, Y \subset Z \Rightarrow X \subset Z$ ”

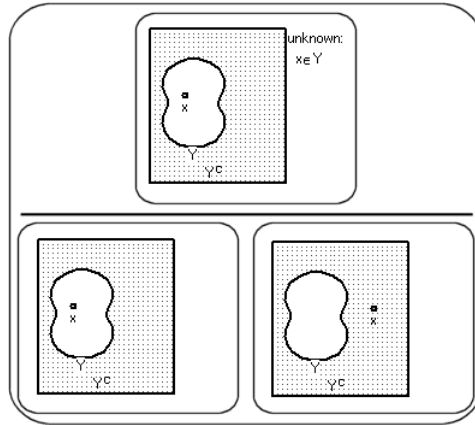
Rule 3. Set equality (an implicit inference rule): “ $X \subset Y, Y \subset X \Rightarrow X = Y$ ”



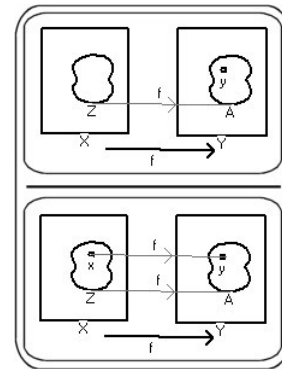
Rule 4. Complement set
- definition 1.



Rule 5. Complement sets exist.

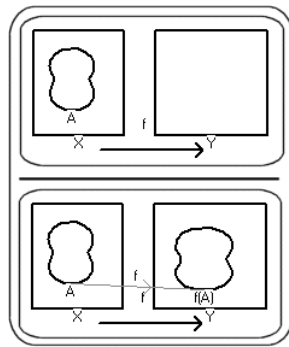


Rule 6. Complement set - definition 2.

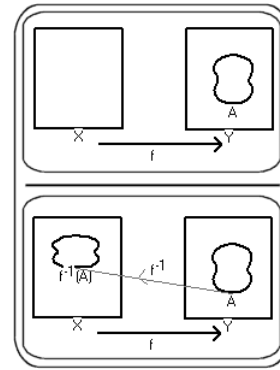


Rule 7. Image set - apply.

Rule 8. Image set – recognise (an implicit inference rule): $x \in X, y = f(x) \Rightarrow y \in f(X)$



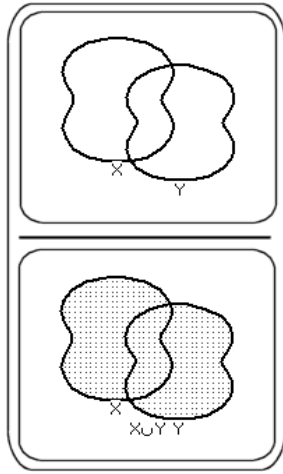
Rule 10. Image sets exist.



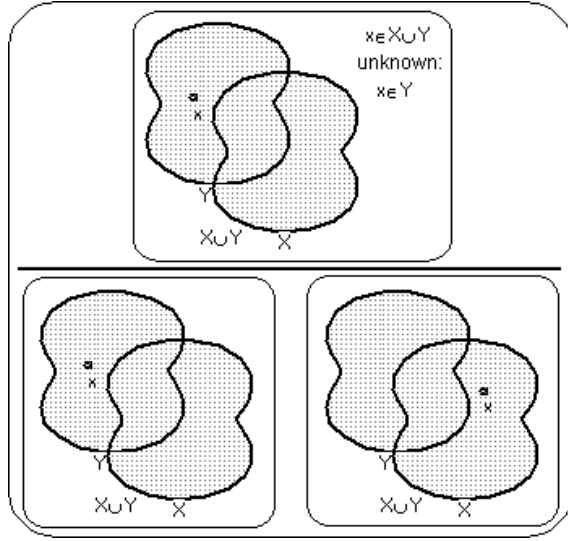
Rule 9. Inverse sets exist.

Rule 11. Inverse set – apply (an implicit inference rule): $x \in X, X = f^{-1}(Y) \Rightarrow f(x) \in f(Y)$

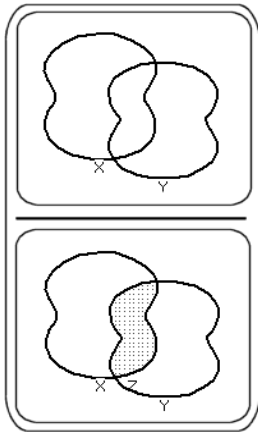
Rule 12. Inverse set – recognise (an implicit inference rule): $f(x) \in Y \Rightarrow x \in f^{-1}(Y)$



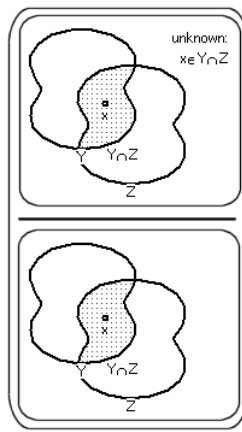
Rule 14. Union sets exist.



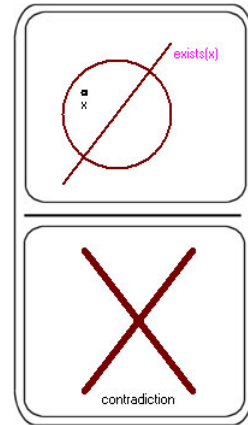
Rule 13. Union set definition.



Rule 15. Intersection sets exist.



Rule 16. Intersection set definition.



Rule 17. Empty set.

Conversions and soundness proofs

Rule 1 (Subset - recognise.): Soundness proof

Let Rule 1 = $R: T_1-s-T_2-f-T_3 \mapsto T'$

Suppose R is not sound.

$\Rightarrow \exists$ reasoning trees $\underline{D}, \underline{D}'$ in EDDLA such that D_0 is consistent, \underline{D} was drawn from D_0 using sound rules Δ , $\underline{D} \xrightarrow{R} \underline{D}'$, but \underline{D}' is inconsistent.

Let $\underline{T} \simeq m \simeq \underline{D}$. Without loss of generality, let the object labels in $m(\underline{T}) \subset \underline{D}$ be the same as those in \underline{T} and assume y' is not used as an object label in \underline{D} .

Let $D = \text{target}(R, \underline{D})$ and $D' = R(D)$.

\underline{D} consistent, \underline{D}' inconsistent $\Rightarrow D$ consistent, D' inconsistent (since all other branches of \underline{D} , if any, are unchanged by the application of R and must therefore have been inconsistent already).

$\underline{D} \xrightarrow{R} \underline{D}' \Rightarrow \text{changes}(D, D') = \{ \emptyset, \{y\}, \{Y \subset X\}, \emptyset \}$ (that is, the universally quantified point y is deleted, and the relation $Y \subset X$ is added to diagram D')

\underline{D} drawn using sound rules $\Rightarrow D$ is consistent. Hence the statement $Y \subset X$ must be inconsistent with D .

D consistent, hence there exist models for D (note: if D is also accurate, then $\text{objects}(D)$ is a model for D). Let $M(D)$ be a model for D with the same object labels. $Y \subset X$ inconsistent with $D \Rightarrow \exists y' \in M(D)$ such that $y' \in Y$ and $\text{not}(y' \in X)$.

Let \underline{A} be the reasoning program $\underline{D}[y'/y]$. But then by Lemma 6, the drawing of \underline{A} is verified by Δ in the same way as for \underline{D} . Hence $\text{target}(R, \underline{A})$ has $y' \in X$.

Δ sound $\Rightarrow \underline{A}$ is consistent, so $y' \in X$.

This is a contradiction, hence R is sound as required.

Rule 2 and Rule 3 are standard definitions.

$\mathbb{I}(\text{Rule 5}) = \text{"euclidean-space}(X), \text{set}(A), A \subset X \Rightarrow \exists 'A^c'. \text{set}('A^c'), 'A^c' \subset X, 'A^c' = A^c\text{"}$

True from the closure of subsets under complementation.

$\mathbb{I}(\text{Rule 4}) = \text{"euclidean-space}(X), \text{set}(Y), \text{set}('Y^c'), 'Y^c' = Y^c, Y \subset X, Y^c \subset X, x \in Y, x \in Y^c \Rightarrow \text{false}\text{"}$

True from the standard definition for Y^c : $x \in Y^c \Rightarrow \text{not}(x \in Y)$

$\mathbb{I}(\text{Rule 6}) = \text{"euclidean-space}(X), \text{set}(Y), \text{set}('Y^c'), 'Y^c' = Y^c, Y \subset X, Y^c \subset X, \text{point}(x), x \in X \Rightarrow x \in Y^c \text{ or } x \in Y\text{"}$

True from $Y \cup Y^c = X$.

$\mathbb{I}(\text{Rule 7}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{set}(A), A \subset Y, \text{set}(Z), Z \subset X, A = f(Z), \text{point}(y), y \in A \Rightarrow \exists x. \text{point}(x), x \in X, f(x) = y\text{"}$

Rule 8: Trivial.

$\mathbb{I}(\text{Rule 10}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{set}(A), A \subset X \Rightarrow \exists 'f(A)'. \text{set}('f(A)'), 'f(A)' \subset Y, 'f(A)' = f(A)\text{"}$

True since $\{y : \exists x \in X, f(x) = y\}$ is a valid set definition.

$\mathbb{I}(\text{Rule 9}) = \text{“euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{set}(A), A \subset Y \Rightarrow \exists 'f^{-1}(A)' . \text{set}('f^{-1}(A)'), 'f^{-1}(A)' \subset X, 'f^{-1}(A)' = f^{-1}(A)\text{”}$

True since $\{x : \exists y \in A, f(x)=y\}$ is a valid set definition.

Rule 11 and Rule 12: Trivial.

$\mathbb{I}(\text{Rule 14}) = \text{“set}(X), \text{set}(Y) \Rightarrow \exists 'X \cup Y' . \text{set}('X \cup Y'), 'X \cup Y' = X \cup Y\text{”}$

True since $\{x : x \in X \text{ or } x \in Y\}$ is a valid set definition.

$\mathbb{I}(\text{Rule 13}) = \text{“set}(X), \text{set}(Y), \text{set}('X \cup Y'), 'X \cup Y' = X \cup Y, \text{point}(x), x \in 'X \cup Y' \Rightarrow x \in X \text{ or } x \in Y\text{”}$

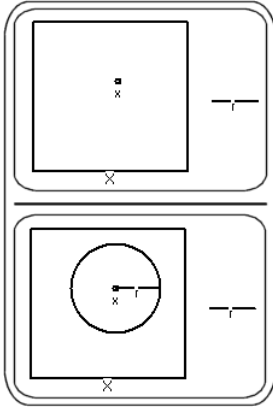
$\mathbb{I}(\text{Rule 15}) = \text{“set}(X), \text{set}(Y) \Rightarrow \exists 'X \cap Y' . \text{set}('X \cap Y'), 'X \cap Y' = X \cap Y\text{”}$

True since $\{x : x \in X, x \in Y\}$ is a valid set definition.

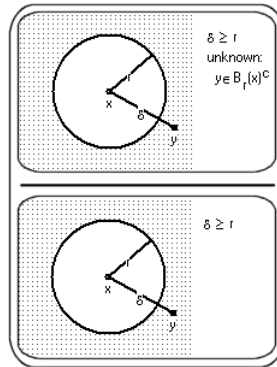
$\mathbb{I}(\text{Rule 16}) = \text{“set}(X), \text{set}(Y), \text{set}('X \cap Y'), 'X \cap Y' = X \cap Y, \text{point}(x), x \in X, x \in Y \Rightarrow x \in 'X \cap Y'\text{”}$

$\mathbb{I}(\text{Rule 17}) = \text{“}\exists x, x \in \emptyset \Rightarrow \text{false”}$

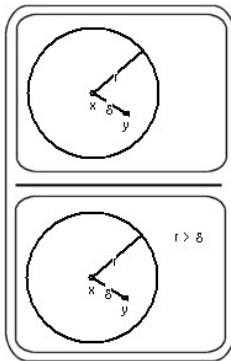
Balls



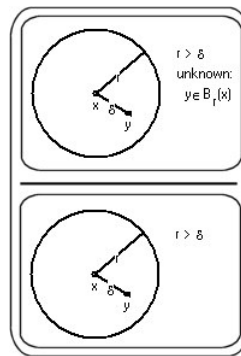
Rule 18. Ball - exists.



Rule 19. Ball - recognise not.



Rule 20. Ball - apply.



Rule 21. Ball - recognise.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule 18}) = \text{"euclidean-space}(X), \text{point}(x), x \in X, \text{line}(r) \Rightarrow \exists B, L. \text{line}(L), L=r, \text{ball}(B), \text{centre}(B,x), \text{radius}(B,L), B \subset X, x \in B\text{"}$

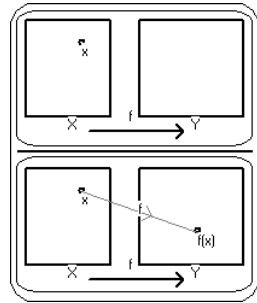
True, since $\text{ball}(B), \text{centre}(B,x), \text{radius}(B,L) \Rightarrow B = \{x' \in X : |x-x'| < L\}$, which is a valid subset of X , and, $x \in B$ is trivial.

$\mathbb{I}(\text{Rule 20}) = \text{"point}(x), \text{point}(y), \text{line}(\delta), \delta=xy, \text{line}(r), \text{ball}(B), \text{centre}(B,x), \text{radius}(B,r), x \in B, y \in B \Rightarrow r > \delta\text{"}$

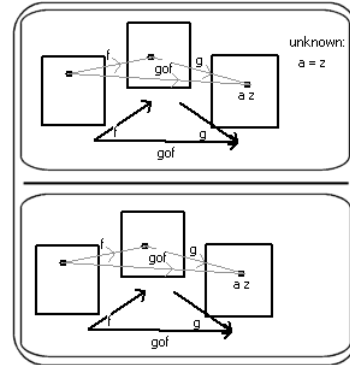
$\mathbb{I}(\text{Rule 21}) = \text{"point}(x), \text{point}(y), \text{line}(\delta), \delta=xy, \text{line}(r), \text{ball}(B), \text{centre}(B,x), \text{radius}(B,r), x \in B, r > \delta \Rightarrow y \in B\text{"}$

$\mathbb{I}(\text{Rule 19}) = \text{"point}(x), \text{point}(y), \text{line}(\delta), \delta=xy, \text{line}(r), \text{ball}(B), \text{centre}(B,x), \text{radius}(B,r), \text{set}('B^c'), 'B^c' = B^c, x \in B, \delta \geq r \Rightarrow y \in 'B^c'\text{"}$

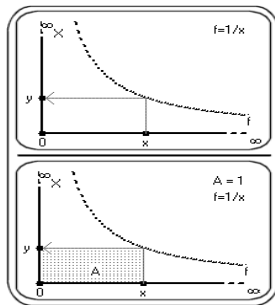
Functions



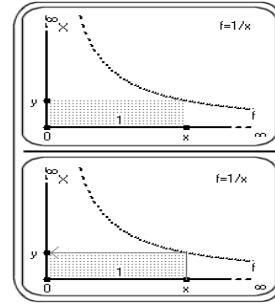
Rule 23. Apply function.



Rule 22. Composed function - apply.



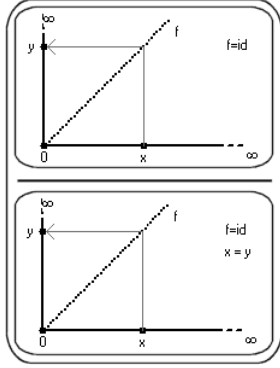
Rule 25. $f(x)=1/x$ - apply.



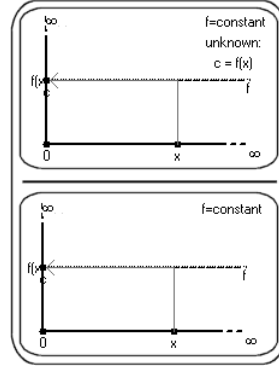
Rule 24. $f(x)=1/x$ - recognise.

Rule 26. function addition: function(f), function(g), function($f+g$), ' $f+g=f+g$ ', point(x), point(y), point($f(x)$), ' $f(x)=f(x)$ ', point($g(x)$), ' $g(x)=g(x)$ ', $y=f+g(x) \Rightarrow y=f(x)+g(x)$ '

Rule 27. function multiplication: function(f), function(g), function($f \cdot g$), ' $f \cdot g=f \cdot g$ ', point(x), point(y), point($f(x)$), ' $f(x)=f(x)$ ', point($g(x)$), ' $g(x)=g(x)$ ', $y=f \cdot g(x) \Rightarrow y=f(x) \cdot g(x)$ '



Rule 28. Identity function
- apply.



Rule 29. Constant function - apply.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule 23}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{point}(x), x \in X \Rightarrow \exists f(x) \cdot \text{point}(f(x)), f(x) \in Y, f(x)=f(x)"}$

$\mathbb{I}(\text{Rule 22}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{euclidean-space}(Z), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{function}(g), \text{domain}(g, Y), \text{range}(g, Z), \text{function}(g \circ f), \text{domain}(g \circ f, X), \text{range}(g \circ f, Z), g \circ f = g \circ f, \text{point}(x), x \in X, \text{point}(y), y \in Y, \text{point}(z), z \in Z, \text{point}(a), a \in Z,$

$y=f(x), z=g(y), a=g \circ f(x) \Rightarrow a=z"$

$\mathbb{I}(\text{Rule 25}) = \text{"1d-set}(X), X=(0, \infty), \text{1d-set}(Y), Y=(0, \infty), \text{function}(f), \text{range}(f, X), \text{domain}(f, Y), f=1/x, \text{point}(x), x \in X, \text{point}(y), y \in Y, y=f(x) \Rightarrow \exists A \cdot \text{area-block}(A), A=x \cdot y, A=1"$

Where $1/x$ is the constant denoting the function $f:(0, \infty) \rightarrow (0, \infty), f(x)=1/x$.

True, since $\forall x \neq 0, x \cdot 1/x = 1$.

$\mathbb{I}(\text{Rule 24}) = \text{"1d-set}(X), X=(0, \infty), \text{1d-set}(Y), Y=(0, \infty), \text{function}(f), \text{range}(f, X), \text{domain}(f, Y), f=1/x, \text{point}(x), x \in X, \text{point}(y), y \in Y, \text{area-block}(1), 1=1 \Rightarrow y=f(x)"}$

True since $x \cdot y = 1 \Rightarrow y = 1/x$.

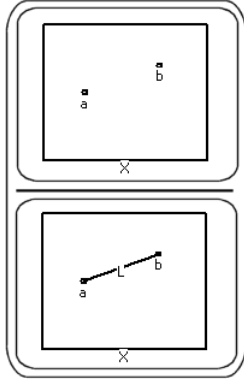
Rule 26: Trivial.

Rule 27: Trivial.

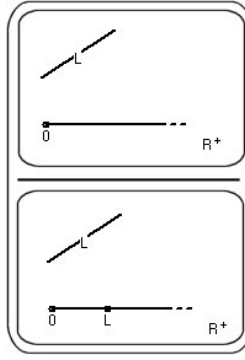
$\mathbb{I}(\text{Rule 28}) = \text{"1d-set}(X), X=(0,\infty), \text{1d-set}(Y), Y=(0,\infty), \text{function}(f), \text{range}(f,X), \text{domain}(f,Y), f=\text{identity}, \text{point}(x), x \in X, \text{point}(y), y \in Y, y=f(x) \Rightarrow y=x\text{"}$

$\mathbb{I}(\text{Rule 29}) = \text{"1d-set}(X), X=(0,\infty), \text{1d-set}(Y), Y=(0,\infty), \text{point}(c), c \in Y, \text{function}(f), \text{range}(f,X), \text{domain}(f,Y), f=\text{constant}(c), \text{point}(x), x \in X, \text{point}(y), y \in Y, y=f(x) \Rightarrow y=c\text{"}$

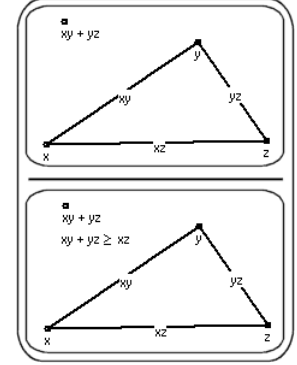
Lines



Rule 30. Line - draw.



Rule 32. Convert line to point.



Rule 31. Triangle rule.

Rule 33. Line equality (an implicit inference rule): “point(a), point(b), line(X), line(Y), $X=ab, Y=ab \Rightarrow X=Y$ ”

Conversions and soundness proofs

We associate lines with the metric function of the space, i.e. if $L=ab$, then L represents $|a-b|$ (which also means $L=ba$). This is implicitly true for the Euclidean spaces we use in drawing, and a visual representational device in other spaces. Note that this association would preclude the use of Euclidean line reasoning (which we have not covered in this project) unless the space is known to be Euclidean.

$\mathbb{I}(\text{Rule 30}) = \text{"euclidean-space}(X), \text{point}(a), \text{point}(b), a \in X, b \in X \Rightarrow \exists L . \text{line}(L), L=ab\text{"}$

True, since X by definition has a metric $(a,b) \rightarrow |a-b|$ that maps any pair of points to a positive number and is symmetric.

$\mathbb{I}(\text{Rule 32}) = \text{"euclidean-space}(X), \text{point}(a), \text{point}(b), a \in X, b \in X \Rightarrow \exists L . \text{line}(L), L=ab=ba\text{"}$

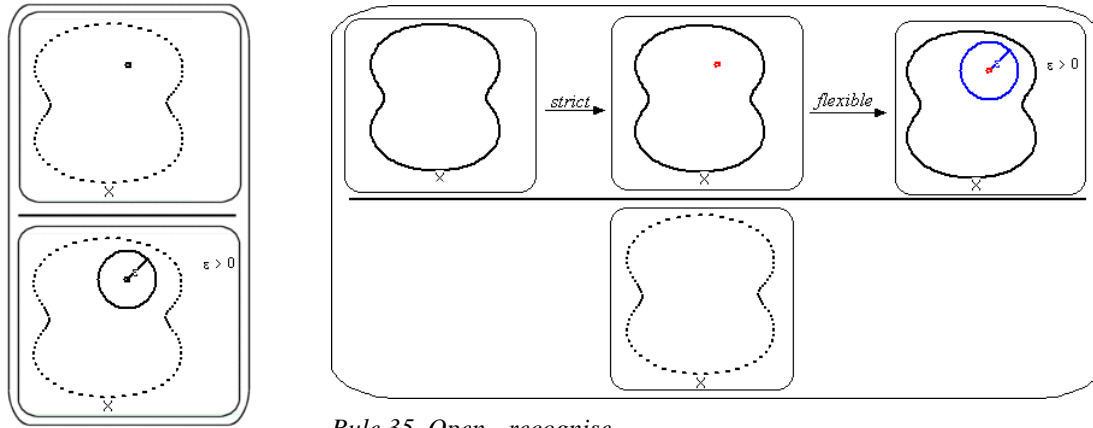
$\mathbb{I}(\text{Rule } 31) = \text{"point}(x), \text{point}(y), \text{point}(z), \text{line}('xy'), \text{line}('yz'), \text{line}('xz'), 'xy'=xy, 'yz'=yz, 'xz'=xz, \text{point}('xy+yz'), 'xy+yz'=xy+yz \Rightarrow 'xy+yz' \geq xz\text{"}$

This is one of the standard axioms for a metric.

Rule 33: Trivial.

12.2.2 Properties

Open



Rule 34. Open - apply.

Rule 35. Open - recognise.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule } 34) = \text{"set}(X), \text{open}(X), \text{point}(x), x \in X \Rightarrow \exists \epsilon, B . \text{line}(\epsilon), \epsilon > 0, \text{ball}(B), \text{centre}(B, x), \text{radius}(B, \epsilon), B \subset X\text{"}$

Rule 35 (Open - recognise.): Soundness proof

Let Rule 35 = $R: T_1 \text{-} s \text{-} T_2 \text{-} f \text{-} T_3 \mapsto T'$

$\mathbb{I}(T_1) = \text{"set}(X)\text{"}$

$\mathbb{I}(T_2) = \text{"set}(X), \text{point}(x), x \in X\text{"}$

$\mathbb{I}(T_3) = \text{"set}(X), \text{point}(x), x \in X, \text{line}(\epsilon), \epsilon > 0, \text{ball}(B), \text{centre}(B, x), \text{radius}(B, \epsilon), x \in B, B \subset X\text{"}$

$\mathbb{I}(T') = \text{"set}(X), \text{open}(X)\text{"}$

Suppose R is not sound.

$\Rightarrow \exists$ reasoning trees $\underline{D}, \underline{D}'$ in EDDLA such that D_0 is consistent, \underline{D} was drawn from D_0 using sound rules Δ , $\underline{D} \xrightarrow{R} \underline{D}'$, but \underline{D}' is inconsistent.

Let $\underline{T} \simeq m \simeq \underline{D}$. Without loss of generality, let the object labels in $m(\underline{T}) \subset \underline{D}$ be the same as those in \underline{T} and assume x', B' are not used as object labels in \underline{D} .

Let $D = \text{target}(R, \underline{D})$ and $D' = R(D)$.

\underline{D} consistent, \underline{D}' inconsistent $\Rightarrow D$ consistent, D' inconsistent (since all other branches of \underline{D} , if any, are unchanged by the application of R and must therefore have been inconsistent already).

$\underline{D} \xrightarrow{R} \underline{D}' \Rightarrow \text{changes}(D, D') = \{ \emptyset, \{x, \varepsilon, B\}, \{\text{open}(X)\}, \emptyset \}$ (that is, the point, the ball and its radial line are deleted, and the relation $\text{open}(X)$ is added to diagram D')

\underline{D} drawn using sound rules $\Rightarrow D$ is consistent. Hence the statement $\text{open}(X)$ must be inconsistent with D .

Let $M(D)$ be a model for D with the same object labels.

$\text{open}(X)$ inconsistent $\Rightarrow \exists x' \in M$ such that $\exists \varepsilon' > 0$, $\text{not}(B_{\varepsilon'}(x') \subset X)$.

Let \underline{A} be the reasoning program $\underline{D}[x'/x]$. But then by Lemma 6, the drawing of \underline{A} is verified by Δ in the same way as for \underline{D} (creating ε'' in the process). Hence

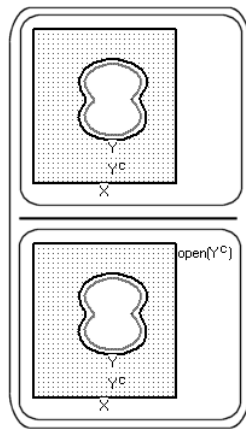
$\text{target}(R, \underline{A})$ has $B_{\varepsilon''}(x') \subset X$, $\varepsilon'' > 0$.

Δ sound $\Rightarrow \underline{A}$ is consistent, so $B_{\varepsilon''}(x') \subset X$.

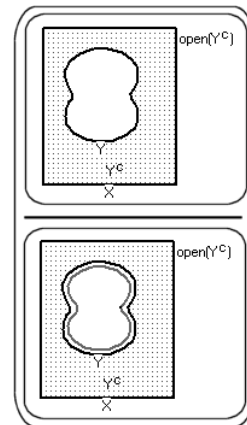
This is a contradiction, hence R is sound as required.

Closed

Because of DDLA's limited sequence reasoning abilities, we use the topological definition of a closed set (which is equivalent to the normal euclidean space definition for euclidean-spaces).



Rule 37. Closed - apply.



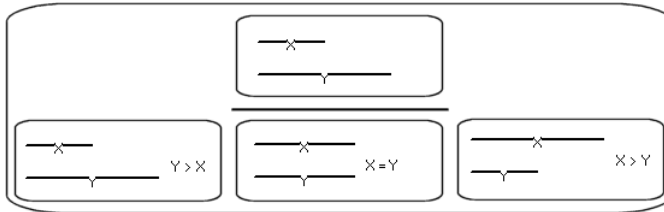
Rule 36. Closed - recognise.

Conversions

$\mathbb{I}(\text{Rule 37}) = \text{"euclidean-space}(X), \text{set}(Y), \text{set}(Y^c), Y^c = Y^c, \text{closed}(Y) \Rightarrow \text{open}(Y^c)"$

$\mathbb{I}(\text{Rule 36}) = \text{"euclidean-space}(X), \text{set}(Y), \text{set}(Y^c), Y^c = Y^c, \text{open}(Y^c) \Rightarrow \text{closed}(Y)"$

Order (<)

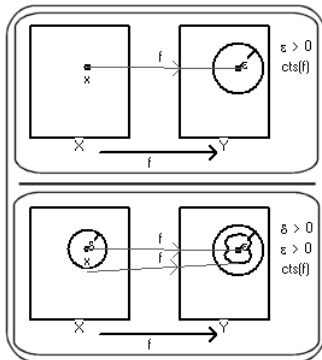


Rule 38. Compare lengths.

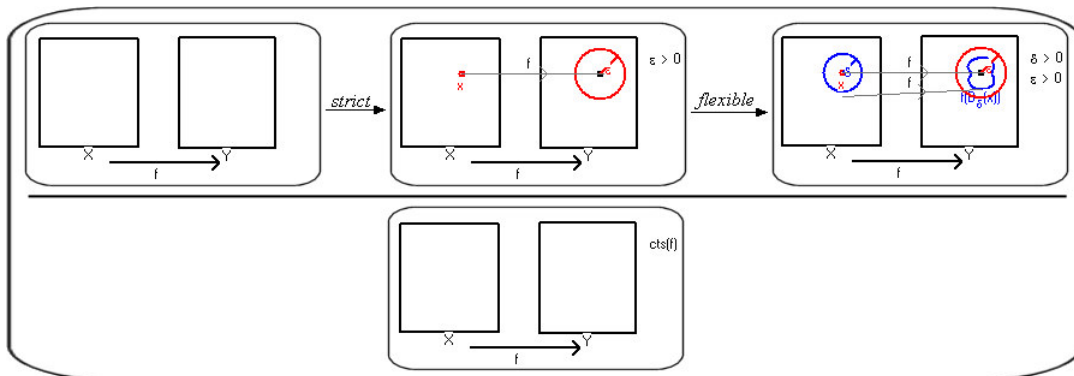
Conversions

$\mathbb{I}(\text{Rule 38}) = \text{"line}(X), \text{line}(Y) \Rightarrow Y > X \text{ or } X = Y \text{ or } X > Y"$

Continuity



Rule 39. Continuity - apply.



Rule 40. Continuity - recognise.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule } 39) = \text{“euclidean-space}(X), \text{ euclidean-space}(Y), \text{ function}(f), \text{ domain}(f,X),$
 $\text{range}(f,Y), \text{ continuous}(f), \text{ point}(x), x \in X, \text{ point}(y), y \in Y, y=f(x), \text{ line}(\epsilon), \epsilon > 0, \text{ ball}(B),$
 $B \subset Y, \text{ centre}(B,y), \text{ radius}(B,\epsilon) \Rightarrow \exists \delta, C, D . \text{ line}(\delta), \delta > 0, \text{ ball}(C), C \subset X, \text{ centre}(C,x),$
 $\text{radius}(C,\delta), \text{ set}(D), D \subset B, D=f(C)\text{”}$

Rule 40 (Continuity - recognise.): Soundness proof

Let Rule 40 = $R: T_1-s-T_2-f-T_3 \mapsto T'$

$\mathbb{I}(T_1) = \text{“euclidean-space}(X), \text{ euclidean-space}(X), \text{ function}(f), \text{ domain}(f,X),$
 $\text{range}(f,Y)\text{”}$

$\mathbb{I}(T_2) = \mathbb{I}(T_1), \text{“point}(x), x \in X, \text{ point}(y), y \in Y, y=f(x), \text{ line}(\epsilon), \epsilon > 0, \text{ ball}(B),$
 $B \subset Y, \text{ centre}(B,y), \text{ radius}(B,\epsilon)\text{”}$

$\mathbb{I}(T_3) = \mathbb{I}(T_2), \text{“line}(\delta), \delta > 0, \text{ ball}(C), C \subset X, \text{ centre}(C,x), \text{ radius}(C,\delta), \text{ set}(f$
 $(C)), f(C) \subset B, f(C)=f(C)\text{”}$

$\mathbb{I}(T') = \mathbb{I}(T_2), \text{“continuous}(f)\text{”}$

Suppose R is not sound. As with Rule 35, this implies \exists reasoning trees $\underline{D}, \underline{D}'$ in $EDDLA$ such that D_0 is consistent, \underline{D} was drawn from D_0 using sound rules $\Delta, \underline{D} \xrightarrow{R} \underline{D}'$, but \underline{D}' is inconsistent.

\Rightarrow the statement $\text{continuous}(f)$ must be inconsistent with D .

Let M be a model for D with the same object labels.

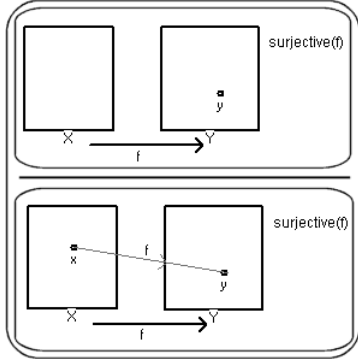
$\text{continuous}(f)$ inconsistent $\Rightarrow \exists x', \epsilon \in M$ (and also, $y'=f(x'), B'=B_\epsilon(y')$) such that
 $x' \in X, \epsilon > 0$ and $\forall \delta' > 0, \text{ not}(f(B_{\delta'}(x')) \subset B')$.

Let \underline{A} be the reasoning program $\underline{D}[x'/x, \epsilon'/\epsilon, y'/y, B'/B]$. But then by Lemma 6, the drawing of \underline{A} is verified by Δ in the same way as for \underline{D} (creating a suitable δ').

Hence $\text{target}(R, \underline{A})$ has $B_{\delta'}(x') \subset B'$.

Δ sound $\Rightarrow \underline{A}$ is consistent, so $B_{\delta'}(x') \subset B'$, which is a contradiction, hence R is sound as required.

Surjective

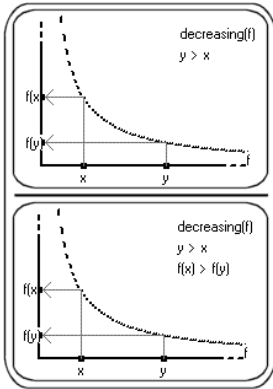


Rule 41. Surjective - apply.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule 41}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f, X), \text{range}(f, Y), \text{surjective}(f), \text{point}(y), y \in Y \Rightarrow \exists x. \text{point}(x), x \in X, f(x) = y\text{"}$

Decreasing



Rule 42. Decreasing - apply.

Conversions and soundness proofs

$\mathbb{I}(\text{Rule 42}) = \text{"1d-set}(X), \text{1d-set}(Y), \text{function}(f), \text{range}(f, X), \text{domain}(f, Y), \text{point}(x), x \in X, \text{point}(y), y \in X, \text{point}(f(x)), f(x) \in Y, \text{point}(f(y)), f(y) \in Y, f(x) = f(x), f(y) = f(y), \text{decreasing}(f), y > x \Rightarrow f(x) > f(y)\text{"}$

Supremum

Rule 43: $\text{1d-set}(R), \text{1d-set}(X), X \subset R, \text{point}(x), x \in R \supset \text{upper_bound}(X, x)$

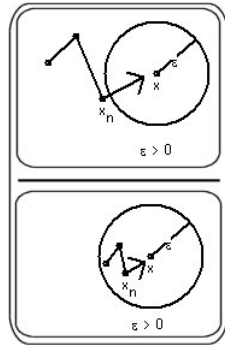
Rule 44: $\text{sup}(X, x), \text{upper_bound}(X, y) \Rightarrow y \geq x$

Rule 45: $\text{1d-set}(X), \text{point}(x), \text{upper_bound}(X, x), \text{point}(y), y \in X \Rightarrow x \geq y$

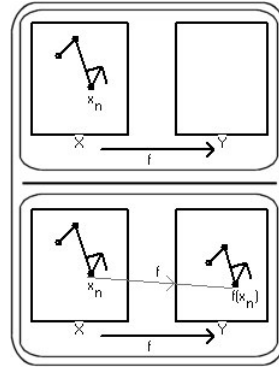
Soundness

Rule 43, Rule 44 and Rule 45 are standard definitions.

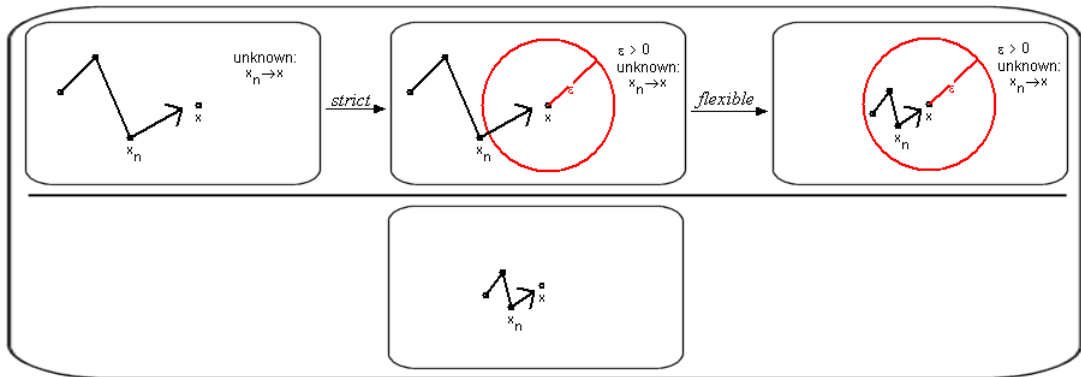
Convergence



Rule 47. Convergent - apply.



Rule 46. Apply function.



Rule 48. Convergent - recognise.

Conversions and soundness proofs

Recall that we interpret relations involving sequences as applying only to the head of the sequence (e.g. for $x_n=1,2,3,4\dots$ we would say $x_n>1$ is true).

$\mathbb{I}(\text{Rule 46}) = \text{"euclidean-space}(X), \text{euclidean-space}(Y), \text{function}(f), \text{domain}(f,X), \text{range}(f,Y), \text{sequence}(x_n), x_n \in X \Rightarrow \exists 'f(x_n)' . \text{sequence}('f(x_n)'), 'f(x_n)' \in Y, f(x_n) = 'f(x_n)'"$

$\mathbb{I}(\text{Rule 47}) = \text{"point}(x), \text{sequence}(x_n), x_n \rightarrow x, \text{line}(\epsilon), \epsilon > 0, \text{ball}(B), \text{centre}(B,x), \text{radius}(B,\epsilon), x \in B \Rightarrow x_n \in B"$

Rule 48 (Convergent - recognise.): Soundness proof

Let Rule 48 = $R:T_1-s-T_2-f-T_3 \mapsto T'$

$\mathbb{I}(T_1) = \text{"point}(x), \text{sequence}(x_n)\text{"}$

$\mathbb{I}(T_2) = \mathbb{I}(T_1), \text{"line}(\epsilon), \epsilon > 0, \text{ball}(B), \text{centre}(B, x), \text{radius}(B, \epsilon), x \in B\text{"}$

$\mathbb{I}(T_3) = \mathbb{I}(T_2), \text{"}x_n \subset B\text{"}$

$\mathbb{I}(T') = \text{"point}(x), \text{sequence}(x_n), x_n \rightarrow x\text{"}$

Suppose R is not sound. As with Rule 35, this implies \exists reasoning trees $\underline{D}, \underline{D}'$ in $EDDLA$ such that D_0 is consistent, \underline{D} was drawn from D_0 using sound rules Δ , $\underline{D} \xrightarrow{R} \underline{D}'$, but \underline{D}' is inconsistent.

\Rightarrow the statement $x_n \rightarrow x$ must be inconsistent with D .

Let M be a model for D with the same object labels.

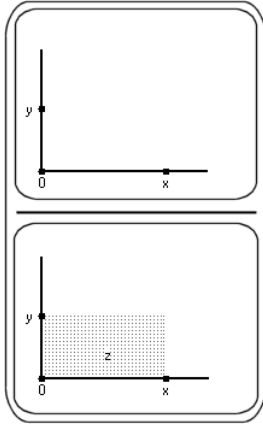
$x_n \rightarrow x$ inconsistent $\Rightarrow \exists \epsilon' > 0, \text{not}(x_n \subset B_{\epsilon'}(x))$

Let \underline{A} be the reasoning program $\underline{D}[\epsilon'/\epsilon]$. But then by Lemma 6 (c.f. §5.2.2), the drawing of \underline{A} is verified by Δ in the same way as for \underline{D} . Hence $\text{target}(R, \underline{A})$ has $x_n \subset B_{\epsilon'}(x)$

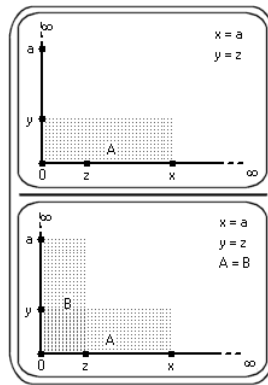
Δ sound $\Rightarrow \underline{A}$ is consistent, so $x_n \subset B_{\epsilon'}(x)$.

This is a contradiction, hence R is sound as required.

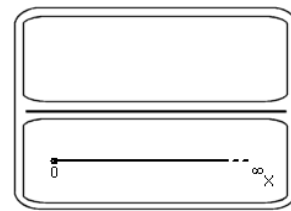
Arithmetic



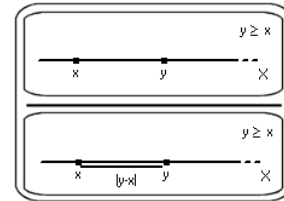
Rule 49. multiply.



Rule 50. multiply –
commutative / rotate
preserves area.



Rule 51. Number line - exists.



Rule 52. subtraction /
Convert point to line.

Rule 53: $(a+b).c = a.c + b.c$

Rule 54: $|x-y| \Rightarrow |x-y|=x-y, x>y$ or $|x-y|=y-x, y<x$ or $|x-y|=0, x=y$

Rule 55: $a+b=c \Rightarrow b+a=c$

Rule 56: $0+a=b \Rightarrow a=b$

Conversions

$\mathbb{I}(\text{Rule 49}) = \text{"1d-set}(X), X=(0,\infty), \text{1d-set}(Y), Y=(0,\infty), \text{point}(x), x \in X, \text{point}(y), y \in Y \Rightarrow$

$\exists A . \text{area-block}(A), A=x.y\text{"}$

$\mathbb{I}(\text{Rule 50}) = \text{"1d-set}(X), X=(0,\infty), \text{1d-set}(Y), Y=(0,\infty), \text{point}(x), x \in X, \text{point}(y), y \in Y \Rightarrow$

$\exists A . \text{area-block}(A), A=x.y, \text{point}(z), z \in X, \text{point}(a), a \in Y, z=y, a=x \Rightarrow \exists B . \text{area-block}(B), B=A\text{"}$

$\mathbb{I}(\text{Rule 52}) = \text{"1d-set}(X), \text{point}(x), x \in X, \text{point}(y), y \in X, y \geq x \Rightarrow \exists 'y-x' . \text{1d-set}('y-x'),$
 $\text{end-points}('y-x', \{x, y\}), y=x+'y-x'\text{"}$

$\mathbb{I}(\text{Rule 51}) = \text{"}\Rightarrow \exists X, x . \text{1d-set}(X), X=(0,\infty), \text{point}(x), x=0\text{"}$

Rule 53 is a standard axiom.

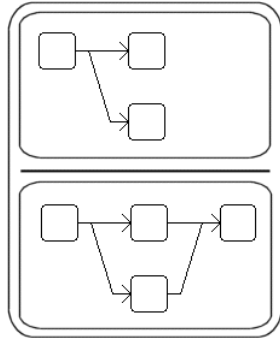
Rule 54 is trivial.

Rule 55 and Rule 56 are standard axioms.

Logic

Rule 57. Exists/not-exists: $"X \Rightarrow \exists X \text{ or } \text{not}(\exists X)"$

Rule 58. Fix object: $"X \Rightarrow \text{specific}(X)"$



Rule 59. Merge cases.

Soundness

Rule 57: Trivial.

Rule 58: c.f. §5.2.2.

Rule 59: c.f. §5.2.2.

Appendix B: Sequence Reasoning

Example

This appendix presents an example proof requiring reasoning with sequences. This proof illustrates the diagrammatic sequence reasoning method proposed in §8.2.5. The theorem we consider is: X closed under sequence convergence $\Rightarrow X^c$ open. Clearly, any proof of this theorem must involve reasoning with sequences.

Our proof will use proof-by-contradiction, with the following two rules to define 'not open':

- 1) $\text{not}(\text{open}(A)) \Rightarrow \exists a \in A . \text{border-point}(a)$
- 2) $a \in A, \text{border-point}(a), B_r(a), r > 0 \Rightarrow \exists b \in B_r(a), b \notin A$

X closed under sequence convergence $\Rightarrow X^c$ open

Proof

Assume false.

$\Rightarrow \exists X$ closed, X^c not open

$\Rightarrow \exists y \in X^c . \text{border-point}(y)$, as shown in Figure 13.1

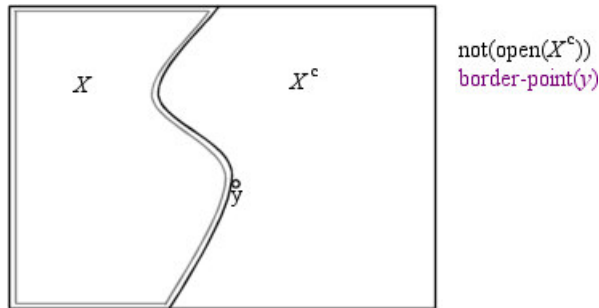


Figure 13.1. Closed X and X^c .

We now construct the first few terms of a sequence, as shown in Figure 13.3.

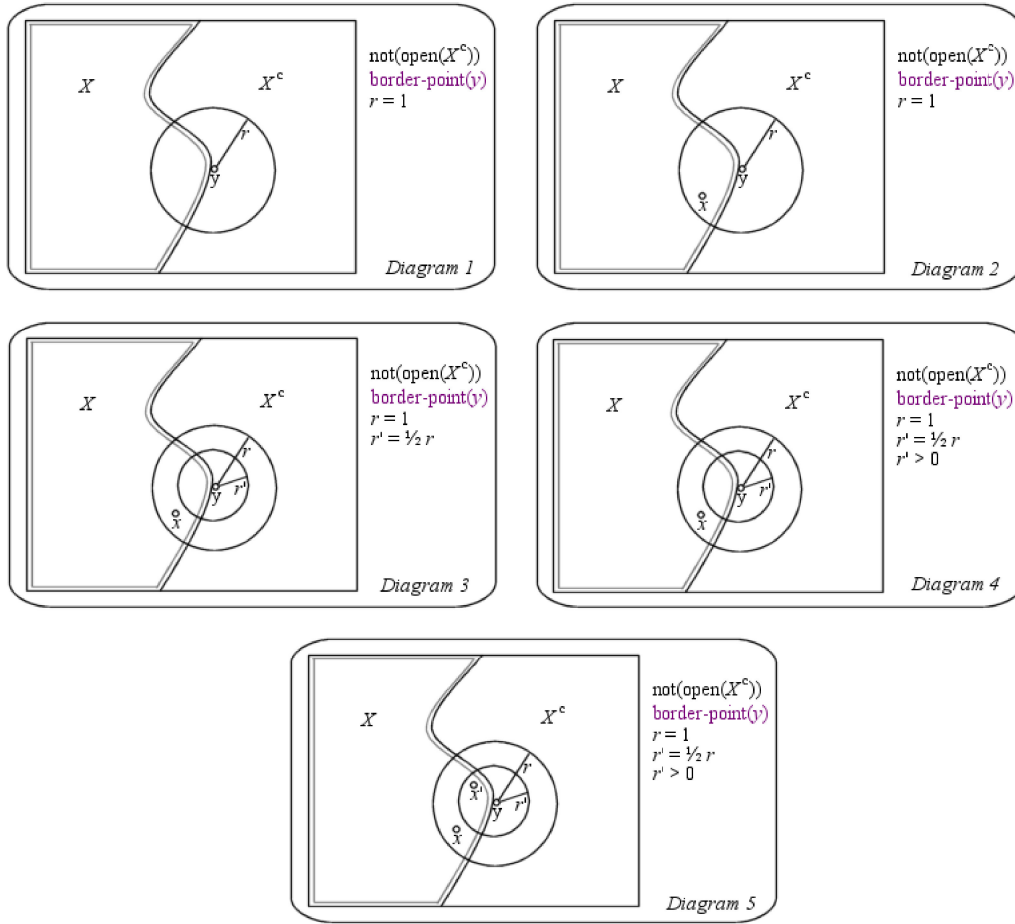


Figure 13.2. Constructing example points in a sequence.

The sequence of steps

- 1) Given a diagram D matching *Diagram 2* above, let $r' = \frac{1}{2}r$ and draw $B_{r'}(y)$
- 2) $r' > 0$
- 3) Find x' such that $x' \in B_{r'}(y)$, $x' \in X$, giving diagram D'

can be repeated indefinitely. This can be verified automatically by checking that $D \simeq D'$ with the 'old objects' mapping to the 'new objects' (i.e. under the mapping $[r \rightarrow r', B_r(y) \rightarrow B_{r'}(y), x \rightarrow x']$). Thus this sequence of steps defines a sequence of lengths, balls and points.

From Figure 13.2, we now extract a 'sequence generator redraw rule' $R: D_n \mapsto D_{n+1}$ (Figure 13.3).

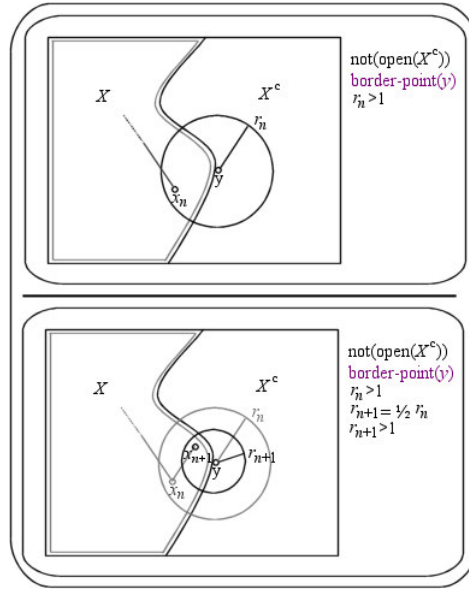


Figure 13.3. Sequence generator rule.

From this, we can see that $x_n \in B_{r_n}(y) \forall n' > n$ (using the transitivity of \subset), and that $r_n = 2^{-n}$ (using a trivial proof by induction).

Thus, using the lemma $\varepsilon > 0 \Rightarrow \exists N \in \mathbb{N} . 2^{-N} < \varepsilon$, we can prove $x_n \rightarrow y$, as shown in Figure 13.4 (c.f. §12.2.2, Rule 48 for the redraw rule definition of convergence).

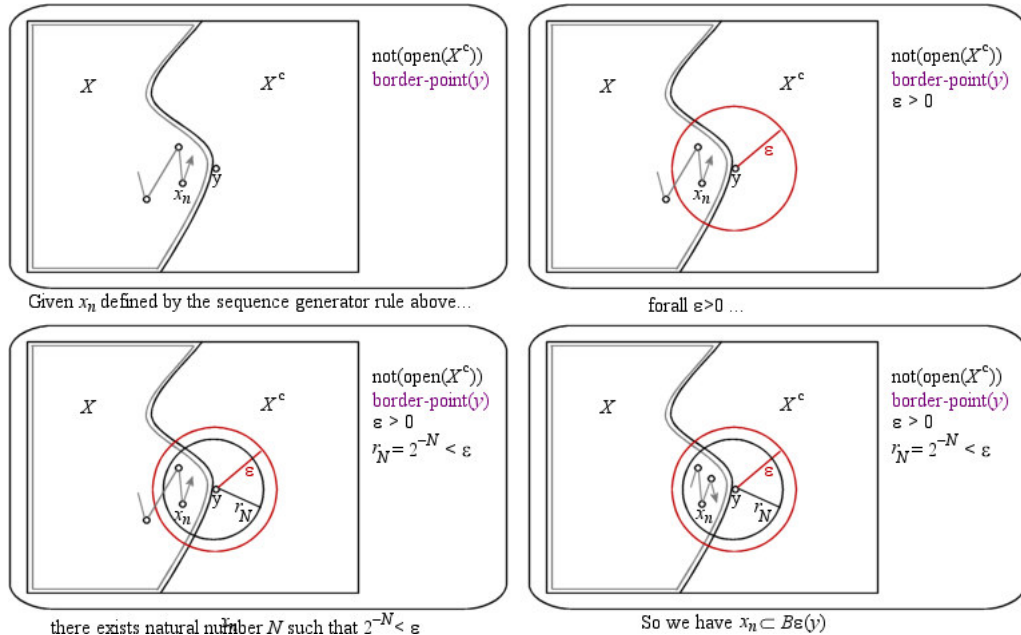


Figure 13.4. Proving the sequence converges to y .

Hence $x_n \rightarrow y$, and X closed under sequence convergence $\Rightarrow y \in X$.

This contradicts $y \in X^c$, so we are done.

Appendix C: Some Example Proofs

This appendix presents three example proofs. §4.1 shows another proof, and a larger set of example proofs is included on the CD-ROM. As discussed in §4.1.1, these proofs are better suited to a dynamic presentation (e.g. on computer) than static paper representations, so we recommend viewing the CD-ROM version where possible.

14.1 Nested balls lemma

This is a short proof for a simple lemma: $\varepsilon \geq \delta \Rightarrow B_\delta(x) \subset B_\varepsilon(x)$, which is stated in Figure 14.1.

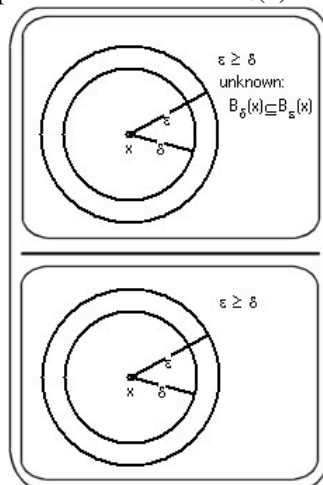


Figure 14.1. Lemma statement.

14.1.1 Proof

The proof has 5 steps, and involves 6 diagrams.

D1: We start with the theorem antecedent.

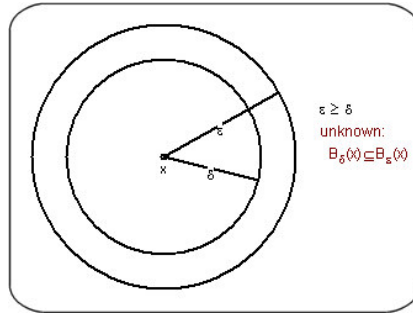


Figure 14.2. D1.

D2: We then apply the “Draw a point” rule (taken from the antecedent of Rule 1) to create an arbitrary point $y \in B_\delta(x)$. We will need to show that $y \in B_\epsilon(x)$ – which is currently marked as unknown.

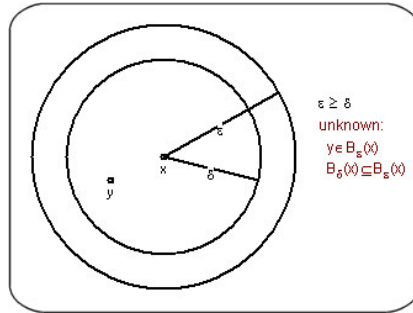


Figure 14.3. D2.

D3: We start by considering the distance from y to x (using Rule 30)

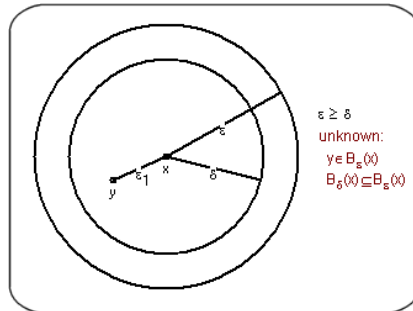


Figure 14.4. D3.

D4: Now $y \in B_\delta(x) \Rightarrow \epsilon_1 = |x-y| < \delta$ (Rule 20, half of the open ball definition).

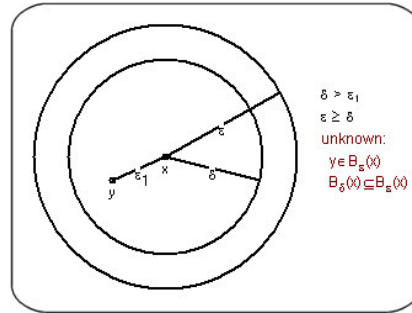


Figure 14.5. D4.

D5: $\varepsilon > |x-y| \Rightarrow y \in B_\varepsilon(x)$ (using Rule 21 – the other half of the ball definition – with $\varepsilon \geq \delta$, $\delta > \varepsilon_1$ to get $\varepsilon > \varepsilon_1 = |x-y|$).

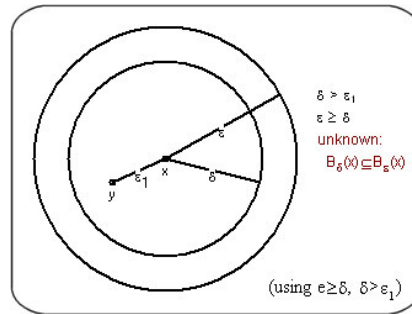


Figure 14.6. D5.

D6: Given an arbitrary point $y \in B_\delta(x)$, we have now shown that $y \in B_\varepsilon(x)$. Hence $B_\delta(x) \subset B_\varepsilon(x)$ (using Rule 1). The theorem consequent will now match diagram D6, so we have reached the end of the proof.

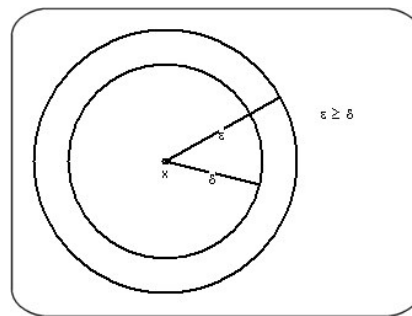


Figure 14.7. D6.

14.2 Open set union

Figure 14.8 shows the theorem statement: Given sets $Y, Z, Y \cup Z$ such that $\text{open}(Y), \text{open}(Z)$ then $\text{open}(Y \cup Z)$.

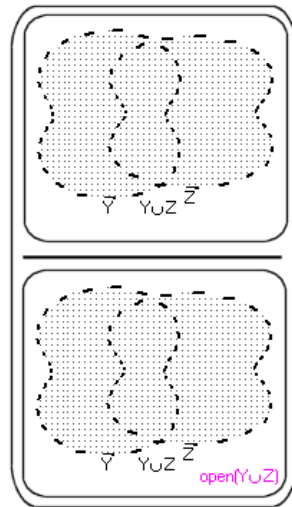


Figure 14.8. The theorem statement.

14.2.1 Proof

Figure 14.9 shows the structure of the proof program for this theorem, which involves a case-split.

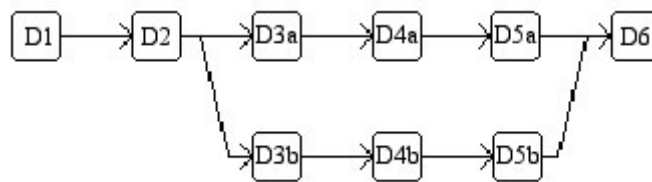


Figure 14.9. The structure of the proof program.

D1: The proof starts with the theorem antecedent.

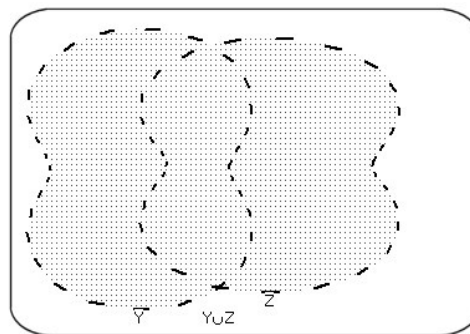


Figure 14.10. Diagram D1.

D2: We draw an arbitrary point in $Y \cup Z$.

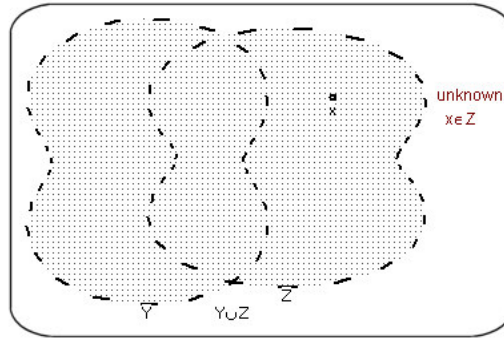


Figure 14.11. Diagram D2.

D3a and D3b: Applying the branch rule “Set union definition”, we split into two cases: $x \in Y$ or $x \in Z$.

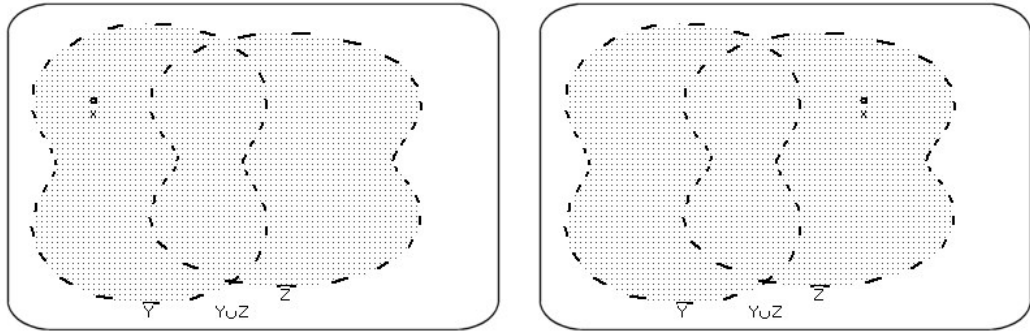


Figure 14.12. Diagram D3a and D3b.

D4a and D4b: We now work separately (but in this proof, identically) on the two cases. In both cases, we can use the rule “Open set – apply definition” to draw balls of radius ε contained in $Y \cup Z$.

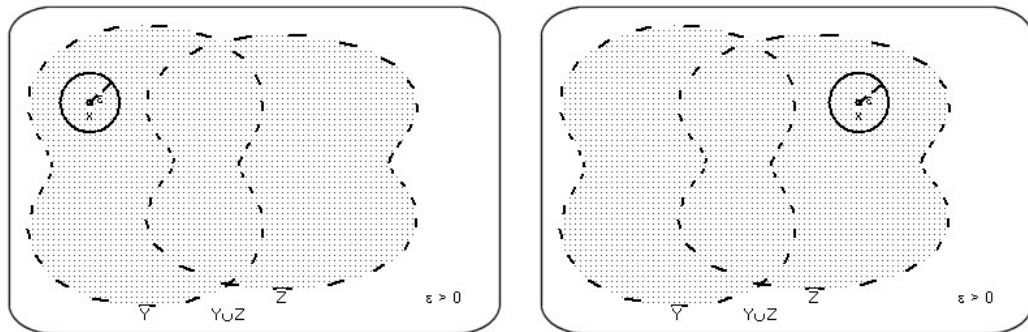


Figure 14.13. Diagram D4a and D4b.

D5a and D5b: In both cases, the rule “Open set – recognise” can now be applied. This is an animated rule, and its antecedent matches diagrams D1, D2, D4a (creating D5a), and D2, D3, D4b (creating D5b).

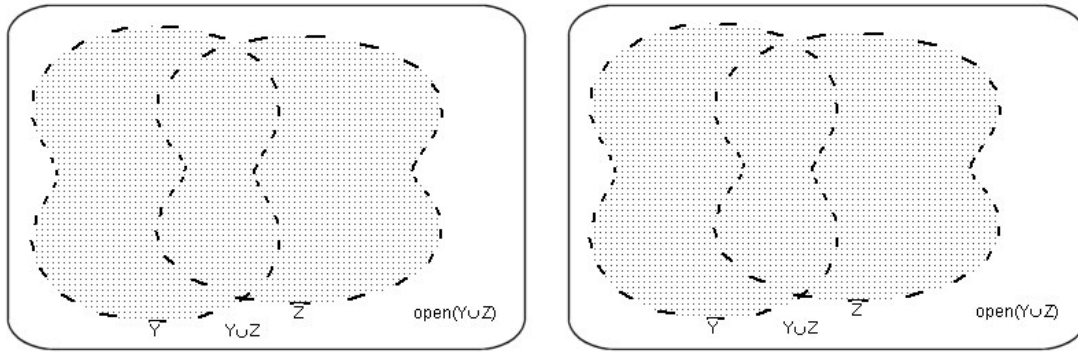


Figure 14.14. Diagram D5a and D5b.

D6: Applying the meta-rule *merge*, we draw the two cases back together to reach the theorem consequent.

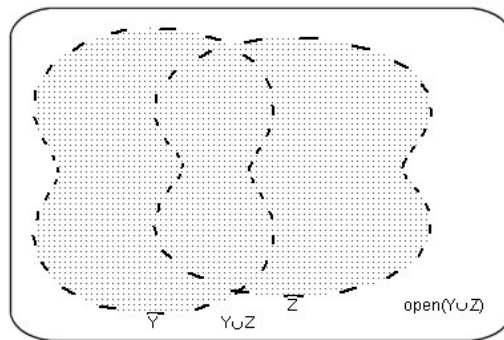


Figure 14.15. Diagram D6.

This completes the proof, as the system will confirm (Figure 14.16).

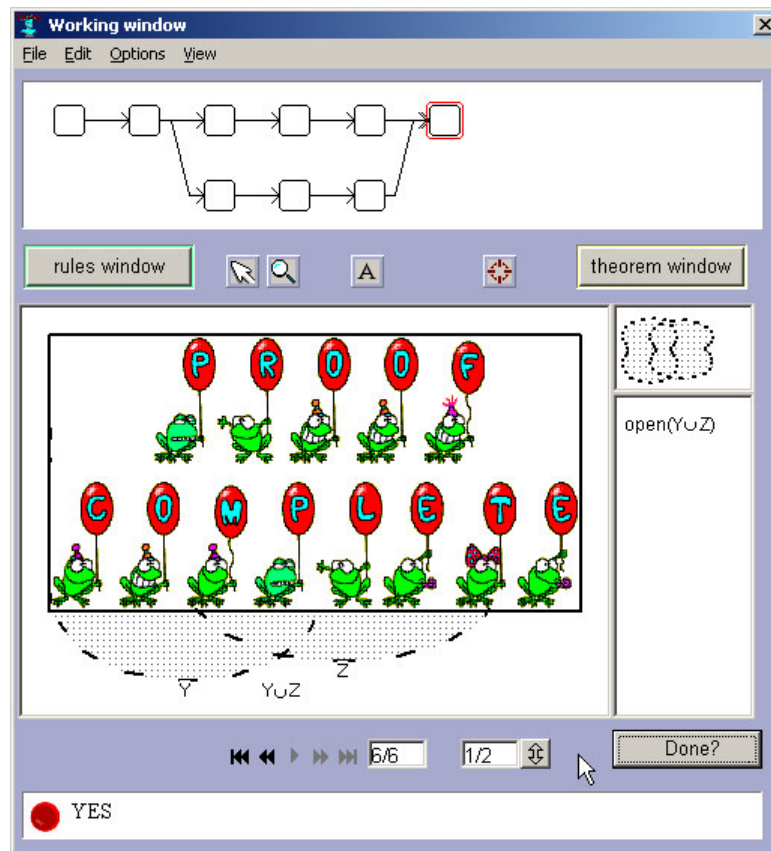


Figure 14.16. Dr.Doodle screenshot showing the verified proof.

14.3 A theorem about continuity

Figure 14.17 shows the theorem statement: Given euclidean spaces X , Y , continuous function $f: X \rightarrow Y$, sets $Z \subset Y$ such that $\text{open}(Z)$, and $f^{-1}(Z) \subset X$, then we have $\text{open}(f^{-1}(Z))$. This theorem can also be stated as “metric space continuity \Rightarrow topological continuity”, given the appropriate definitions.

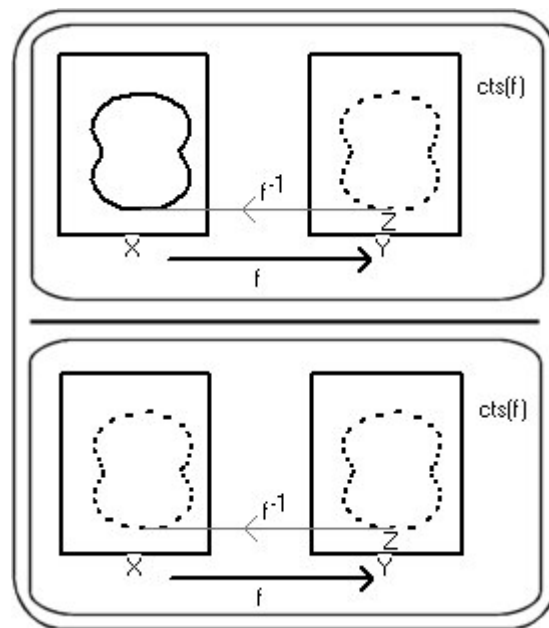


Figure 14.17. The theorem statement.

14.3.1 Proof

The proof involves a chain of 10 diagrams.

D1: The proof starts with the theorem antecedent.

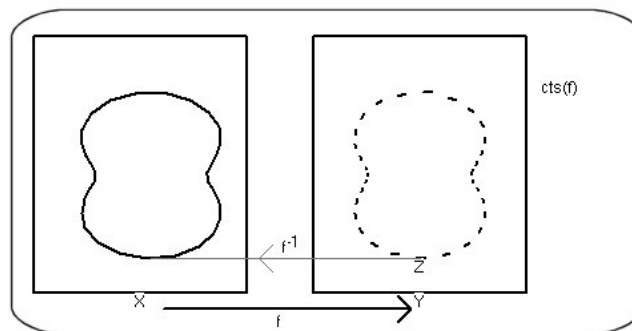


Figure 14.18. Diagram D1.

D2: We draw an arbitrary point in $f^{-1}(Z)$.

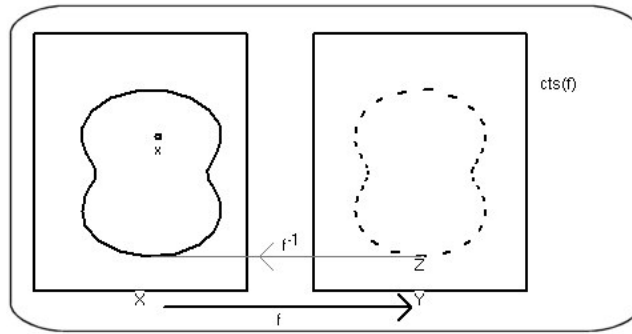


Figure 14.19. Diagram D2.

D3: Apply f to x to get $y \in Y$, $y = f(x)$. An implicit inference gives us $y \in Z$ (since $x \in f^{-1}(Z)$)

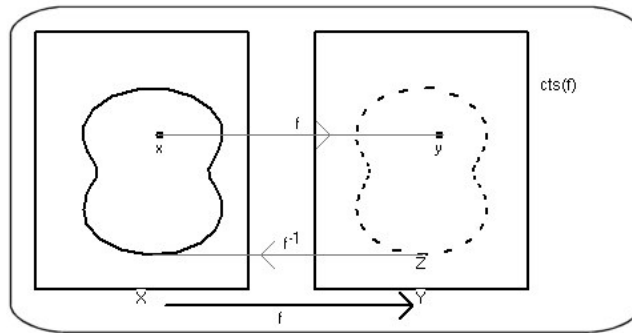


Figure 14.20. Diagram D3.

D4: Y is open, hence we can use 'Open set – apply definition' to create an ϵ -ball inside Y .

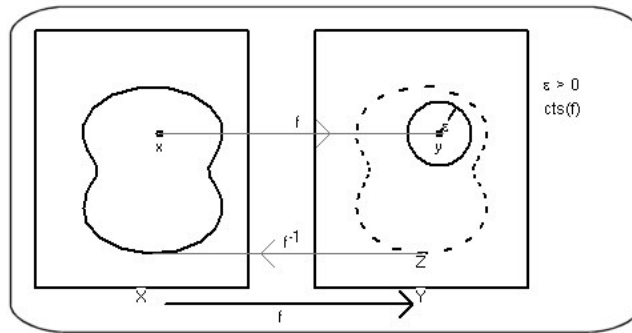


Figure 14.21. Diagram D4.

D5: We now apply the rule defining continuous functions to find a δ -ball about x in X such that $f(B_\delta(x)) \subset B_\epsilon(f(x))$. We draw this δ -ball inside $f^{-1}(Z)$ – but this relation has not been proved yet, so the system adds an appropriate unknown statement.

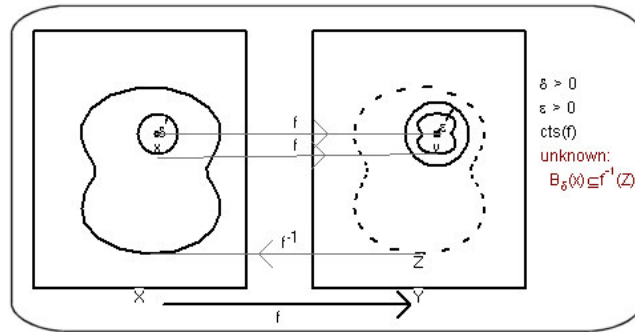


Figure 14.22. Diagram D5.

D6: We want to show that $B_\delta(x)$ is indeed always inside $f^{-1}(Z)$ (note how the unknown statement helps guide the proof here), which we do by showing each point of $B_\delta(x)$ is a member of $f^{-1}(Z)$. The first step is to consider an arbitrary point $z \in B_\delta(x)$.

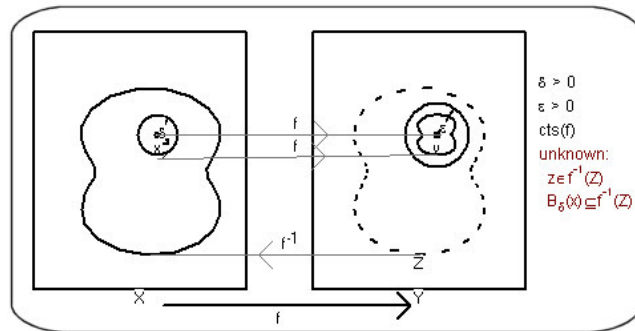


Figure 14.23. Diagram D6.

D7: Then apply f to z , creating a point inside $f(B_\delta(x))$.

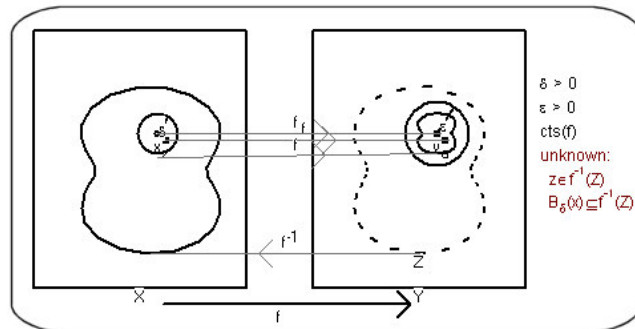


Figure 14.24. Diagram D7.

D8: We now use the inverse function definition to say that $f(z) \in Z \Rightarrow z \in f^{-1}(Z)$, eliminating one of our unknown statements.

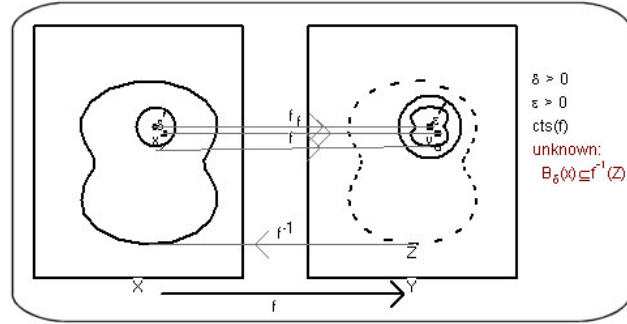


Figure 14.25. Diagram D8.

D9: We have now shown that any point in $B_{\delta}(x)$ is also in $f^{-1}(Z)$. The animated rule “Recognise set inside” will match diagrams D5, D6 and D8, adding the relation $B_{\delta}(x) \subseteq f^{-1}(Z)$ (i.e. removing the 'unknown' statement).

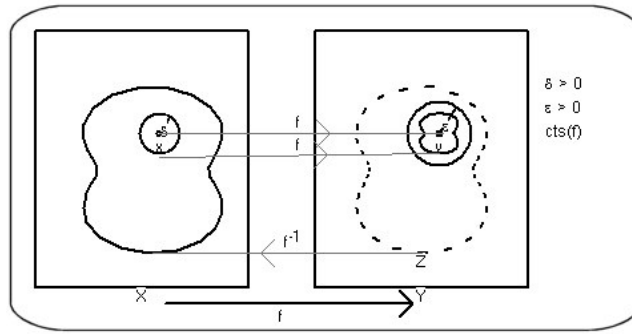


Figure 14.26. Diagram D9

D10: We have now constructed a ball inside $f^{-1}(Z)$ around the arbitrary point x . This means we can apply the “Recognise open set” rule (matching diagrams D1, D2 and D9) to deduce that $f^{-1}(Z)$ is open as required. This concludes the proof.

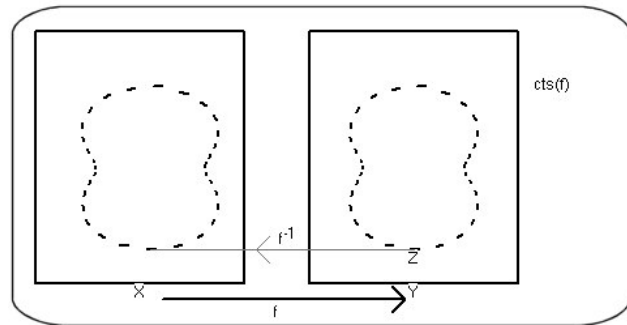


Figure 14.27. Diagram D10.